Spring 2015

Teaching introductory game development with unreal engine: Challenges, strategies, and experiences

Nicholas A. Head
Purdue University

Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Nicholas Head

Entitled

TEACHING INTRODUCTORY GAME DEVELOPMENT WITH UNREAL ENGINE: CHALLENGES, STRATEGIES, AND EXPERIENCES

For the degree of Master of Science

Is approved by the final examining committee:

David Whittinghill
Chair

Sean Brophy

Esteban Garcia

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): David Whittinghill

Approved by: Mihaela Vorvoreanu 4/21/2015

Head of the Departmental Graduate Program Date

TEACHING INTRODUCTORY GAME DEVELOPMENT WITH UNREAL ENGINE: CHALLENGES, STRATEGIES, AND EXPERIENCES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Nicholas A. Head

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2015

Purdue University

West Lafayette, Indiana

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

ABSTRACT

Head, Nicholas A. M.S., Purdue University, May 2015. Teaching Introductory Game Development with Unreal Engine: Challenges, Strategies, and Experiences. Major Professor: David Whittinghill.

From the days of Pong to 100 million dollar projects such as the Grand Theft Auto franchise, video games have evolved significantly over the years. This evolution has also changed the way game development is viewed as a career. Today, video games are one of the most profitable forms of entertainment, and game development courses are appearing at universities around the world. Even with this growth, a degree from a university has yet to be an important factor in finding a job in game development (Owen, 2013). This thesis examines a method of creating and implementing an introductory gaming course and recommends ways to improve the curriculum.

The main focus of the course was to introduce game development to the students. Each week, they were given an exercise that covered a different topic. Students also took part in a team project in which they were tasked with creating a complete game. The goal of the team projects was to expand the student's basic knowledge given to them from the exercises. Data was gathered on the students' subjective experiences with the class. This data and the class's overall performance were compared with past iterations of the course. New to the course was the Unreal Engine. Students used the latest version of the engine, Unreal Engine 4, to complete exercises. Not all students chose to use this engine for the

team project. Instructor and students experiences with the engine were also recorded. While there were some problems implementing the engine within our lab environment, we were still able to execute the overall lesson plan. Even with the engine issues, the course had overall good performance. CGT 241, Introduction to 3D Animation, was shown to help the students to complete the course while CGT 215, Computer Graphics Programming I, did not provide enough information on game programming. Exercises were found to be helpful but students wanted a better understanding of how these skills can be applied to game development. Team projects also went well with most teams creating a functional project. Students wanted more time to complete projects along with a structured approach to the project. Confidence in game development and the Unreal Engine were not high but students were enthusiastic in continuing in the field of game development.

Recommendations were made to the curriculum in order to fix some of the issues with the introductory course and help students find a career. In order to fix the gap between the programming course and the introductory game course, a video game programming course was recommended that focused on teaching students how code works with video game engines. An option to specialize was also recommended in order to see a higher level of understanding on game concepts and a higher level of quality of game projects. Changes to the higher courses were also made for a yearlong course where students would focus on a single project to publish. This would expand on the introductory course while also replicating the game development process.

CHAPTER 1. INTRODUCTION

The goal of the Computer Graphics Technology (CGT) department at Purdue University is to give students the tools to turn ideas into models, digital animations, interactive games, and other disciplines (Computer Graphics Technology, n.d.). At the time of publication, video game courses had been growing steadily in the department for the past several years. The curriculum had grown from a single application development course to Introduction to Game Development, CGT 345, and an advanced course, CGT 445. CGT 345 builds upon concepts from the department's introductory 3D animation courses as well as the programming courses while CGT 445 builds upon CGT 345 with advanced topics such as path finding, artificial intelligence (AI), and game psychology. This paper focuses on the implementation and assessment of the introductory course, CGT 345.

1.1 Research Question

Is CGT 345 successful in teaching the basic skills needed to advance to the next level of game development?

1.2 Statement of Purpose

This thesis evaluates previous and current attempts to teaching introductory game development to undergraduate students. Previous courses have discussed the teaching process of art, programming, and design for game development. The current course's goal is to teach students the basics skills to continue in game development. These skills are a basic understanding of programming, art, and design. This implementation is examined through the instructor and student points of view.

1.3 Scope

This research focuses solely on the introductory game development course at Purdue University. The research assesses the overall performance of the course from the students' and instructor's perspective. Student data were collected using a post survey that explored their experiences in the course. A comparison with the student data was made from the instructor's opinion of the course. Overall, the goal was to see if the course was successful in teaching students the basic skills to advance in game development.

1.4 Significance

Game development courses are a young curriculum that is just starting to take form around the world. It has grown greatly from its computer science origins to be taught from within a variety of disciplines. However, the core components for teaching game development at the college level remain a moving target. Even after completing these courses, a degree in game development has yet to become a standard in the video game

industry. According to recent data, on average a student with only a high school diploma earns more than a student with a bachelor's degree (Graft, 2014). Students are sometimes not even taught industry standard techniques in college courses. This knowledge gap makes it harder to find a game development job and can lead to a large amount of on-the-job training. Game development jobs have been found to place a high emphasis on skills rather than education (McGill, 2008). In personal correspondence, even the author has been told by professional game developers that college is "a waste of time" in the industry. There is a need to improve the current system of teaching students in order to better prepare students for a career in game development. By improving this system, students will be more likely to find jobs and create better games thanks to the skills taught to them at a university.

1.5 Assumptions

This research was performed and conclusions have been drawn using the following assumptions:

- Participants are either enrolled or instructing the introductory course.
- Participants completed the preparatory coursework
- Participants answered all questions of the survey truthfully
- Courses evaluated were related to art, programming, or design principles of game development

1.6 Limitations

This research was limited by the following:

- Subjects were limited to students that enrolled in the Fall 2014 CGT 345 introductory game development course at Purdue University.
- Reviewed only the introductory game development course at Purdue University.
- Did not review game development courses relating to business or human resources.

1.7 Delimitations

This research was performed acknowledging the following delimitations:

- The survey analyzed the overall performance of the student in the course.
- All participants were over the age of 18.

1.8 Definition

AAA Game: Generally a title developed by a large studio or funded by a large budget (Schultz, n.d.).

Game Development: the process of creating video games.

Normal: An outward directional vector that is perpendicular to the surface of a three-dimensional model (Slick, n.d.).

Unreal Engine: A video game engine created by Epic Games.

UV: A two-dimensional plane that represents vertices on a three-dimensional object (Slick, n.d.).

Video Game Engine: a tool that brings together art, programming, and design with the goal of creating a video game (Ward, 2008).

WASD: the computer keyboard keys W, A, S, and D.

1.9 Summary

This chapter introduced the key concepts by going over the purpose, scope, and significance of the study. It also reviewed the assumptions, limitations, and definitions. Chapter will evaluate previous attempts and methods of teaching video game courses.

CHAPTER 2. LITERATURE REVIEW

Chapter 2 reviews game development courses in order to understand how these courses are taught. Understanding the history will reveal what works and where there is room for improvement.

2.1 Background

CGT 345 has gone through different forms and names. The course started as CGT 245 and it focused on mobile application development using Corona SDK. This engine allowed students to develop applications or games for the course. This was before the game development focus had grown so students were generally from the web application field. Similar to the present class, students would complete exercises every week and along a team project. In contrast to the present class, there was also an overall class project where the entire class would coordinate together to create a game. This class wide project was hard to implement and gave students a large amount of work for a single course. The course shifted away from mobile development in 2011 in favor of a studio environment using Unity 3D.

At the time, Unity 3D was a new game engine that was quickly gaining favor with independent developers and smaller studios. It was similar enough to other major 3D engines that it allowed students to create high level projects while it also allowed them to

carry over basic game development principles to other engines. Also, students had access to a free version of Unity 3D that could be installed on their personal computers. A problem arose with this change as some students were unable to complete the modeling exercise. Students used Autodesk Maya, a 3D modeling software, to create a house along with other game objects. The problem was that students had either yet to take the 3D animation course, CGT 241, or they were of another discipline, such as web development. To combat this, the exercise was split into three parts and students were given a step by step tutorial on modeling, unwrapping, and texturing the house. In the end, these changes did not alleviate the problem as students still did not have the basic modeling knowledge needed to complete the exercises. The course was then changed to CGT 345 and the modeling course was added as a prerequisite. One of the outcomes this paper will observe is whether these problems are still occurring in the current course model.

CGT 241 is a survey course for animation. The goal of the course is to prepare students for further study in higher level topics of 3D animation. Course work consists of modeling, animating, texturing, rendering, lighting, and rigging 3D objects. While the class teaches a variety of topics, it gives students enough experience in modeling to create basic shapes or structures. One thing the class does not cover is implementing assets into a video game engine. Certain steps and procedures are different when creating models for a video game in contrast to a 3D animation. However, the modeling differences for the introductory game development course are minor enough to be shown in a demonstration. Currently, if a student was to improve on modeling aspects in the game development field, they need to do this outside of the curriculum.

Another prerequisite is CGT 215, Computer Graphics Programming I. This course focuses on scripting and programming fundamentals, logic, and problem solving. It also provides the basis for developing object-oriented applications by understanding how to write, compile, build, and debug an application. All of this has provided students with enough programming knowledge to complete the exercises in CGT 345 in the past. The course also uses C++ as the main programming language, which is the same coding language as Unreal Engine (unlike Unity that uses C#). The goal of this prerequisite is to prepare students for the programming tasks that they face in the introductory course.

The Unreal Engine has been used by many developers as a basis to turn ideas into games. Titles such as *Mass Effect, Gears of War*, *Bioshock,* and many other successful franchises have used this engine to create engaging and dazzling experiences (Unreal Engine 3, n.d.). In the past, Epic Games, the creator of the Unreal Engine, had made a free version of the engine, known as Unreal Development Kit. It featured a similar interface as Unreal 3 but without some of its major features. In order to get these features, a license would need to be purchased, which could cost upwards of a million of dollars for developers and educators. This steep price gap was one of the reasons that made the Unity 3D engine appealing. The Unreal Engine 4 changed this with the introduction of a subscription based model, $19.95 a month, which offered all of its features with Unreal Engine 4 and could be installed on any number of the lab computers. In contrast, a license was needed for every computer that Unity was installed. If the subscription was canceled, then the owner would still be able to use the version of the engine they owned before the cancellation. Students would only need to pay one payment in order to download the engine for the course. However, they would not be able to update the engine without a

subscription. Also, if the student was going to sell a game they created using the engine, they would need to pay for a subscription and give 5% of the profits to Epic Games. Not only is this a better solution for the department but it also gives students experience with an industry standard engine that can improve their chances of finding an industry job after college.

2.2 Literature Review

This section discusses an approach that will be used to teach the current introductory game development course while also exploring how other programs have implemented their game development courses.

2.2.1 Problem Based Learning

The course was developed around a problem based learning method. Problem based learning is a teaching method that combines the acquisition of knowledge with the development of generic skills (Wood, 2008). Graff and Kolmos (2007) also define it as a method to organize the learning process in a manner that the students are actively engaged in finding the solution. This method started in the 1960's among medical educators that were striving to provide an education in professional practices by engaging students with real problems that doctors encountered (Pease & Kuhn, 2011). Since then, the process has been evolving to affect the education of scientific fields and even game development.

BMU, Belgrade Metropolitan University, used a problem based learning method for their game development course (Timcenko & Stojic, 2012). The later version of the

course even features Autodesk Maya for modelling and the Unity 3D game engine. Similar to CGT 345, students would learn core techniques with a traditional teaching model but also be assigned group problem solving exercises and projects. Instructors would help out with these group assignments but only as an advisor. This thesis differs in that its uses a lab oriented course with students that are less experienced in creating games.

Another example had a sixth grade class work on a sustainable game project (Hwang, Hung, & Chen, 2013). Students would work on a game for 50 minutes a week over a 10 week period to promote a pollution free town. Teachers did not tell the students how to achieve this goal as they just advised and gave guidelines for the project. Instead, students would evaluate each other's work and assess the quality of the project. The data showed that this approach improved students learning achievements, motivation, and problem solving skills in comparison to the conventional game approach. While the researcher will be using a college level course with students creating their own assets, this thesis illustrates the importance of the problem based learning method when used to develop games.

2.2.2 Game Development Courses

Gaming courses have been used for a variety of reasons at the university level. A large amount of these programs started in the Computer Science department. Video games have been used to draw attention and gain applicants for the department. One university tried to create a game based programming course using XNA (Sung, Rosenberg, Panitz, & Anderson, 2008). XNA is a game engine that allows students to

create a console game, Xbox 360, using the C# programming language. Similar to Unreal, XNA is able to handle 2D and 3D objects. However the class only focused on programming, which led them to a 2D approach. In contrast, this paper focuses on a general approach to accommodate for both artistic and mathematically minded students. The program experienced problems adjusting the difficulty of the course due to some assignments and technical issues. The goal of the course was not teach the basics of game development but rather to introduce students to the computer science program.

Another CS course also evaluated its effectiveness in keeping students using game development courses. They found that they were capable of keeping a 93% retention rate through this game development course (Bayliss, 2009). However, students were not able to explore and create their own game until later into the game development program. The paper did highlight the need for some creativity in the class to keep students interested and it highlighted problems such as program issues and time constraints that were also observed in the application of the introductory game development course.

Anderson and McLoughlin (2007) tried to create a class using C-Sheep to teach programming to animation students. The C-Sheep system was a library written in ANSI C that allowed the user to tweak the environment and characters that were provided. Instead of teaching the basics of game development, the course was teaching students the basics of C programming. Along with the different focus, the class was unable to accommodate a team project to explore the principles that they learned in the course. This lack of exploration led to students questioning how the system worked without having instances to try them.

Another way for a program to avoid overloading students with asset creation and programming is to use Mods. Mods are modifications done to existing games usually through an editor provided by a developer. One study used Mod's in a workshop setting to teach students about game development (El-Nasr & Smith, 2006). The two workshops catered for a wide variety of disciplines similar to this thesis. The classroom part of the study used Wildtangent's Web Driver and Unreal Tournament 2003, which gives students access to the Unreal Development Kit. With UDK, they were asked to edit a DeathMatch map using the Unreal Editor, map editor, and Unreal Script. While UDK gave students a taste of the Unreal Engine, Unreal Engine 4 has all of the high end development features available to the students. Similar to our introductory course, the Web Driver exercises gave students step by step instructions on how to create a game. Another similarity is that students were allowed to complete their final project in whatever engine that they chose to use.

Wynters (2007) also used Unreal Tournament except with the focus of teaching non programming principles such as modeling and lighting. Students used modeling software to create models and terrain for the levels. This course was similar to the previous iteration of CGT 345 as it had to teach students how to use the modeling software. CGT 345 has to teach a wider variety of disciplines in comparison to this papers art perspective. Teaching students only about art would take time out of the other core concepts of game development that need to be taught before a student can advance to a higher level course. However, this class was able to teach artistically oriented students about game development.

One course had mixed disciplines that worked together to create a game project (Gestwicki, Sun, & Dean, 2008). Programming and design students worked together to create a single game project over the semester. The first five weeks consisted of the creation of the design document while the last 10 were developing the game. Unlike our course, they had milestones every three weeks instead of every week. They also started around three weeks before our students created their groups. In this course, design students seemed to only be doing conceptual work while the programming students implemented the entire game. In our course, students will still get to experience the different disciplines of game development while also learning a new engine.

The University of Santa Cruz used Game Maker to teach a large number of students, 172 students, about game design (Whitehead, 2008). Game Maker is a 2D game engine that comes with its own built in scripts. Unlike Game Maker, Unreal is able to support both 2D and 3D game projects with the use of outside code. The course features a similar capstone or final project as this paper but the class focuses more on game analysis. Students would analyze and come up with game ideas by playing video games. While it is an interesting concept, it strays away from teaching students the basics of development in order to create a game. At Purdue, students do not dwell into the game design process until they understand how a game is created. While the students learn important skills regarding game design, they do not gain the technical skills needed in order to understand how video games are created.

DePaul University's introductory course also uses Game Maker but their advanced class uses XNA (Linhoff & Settle, 2008). Their course is very similar to Purdue in that students complete exercises and a team project throughout their time in the course.

However, this course design is implemented mostly at the advanced level instead of the introductory level as seen with CGT 345.

Another study used Metaio Creator to teach students about augmented reality (Wichrowski, 2013). This project mixed art and computer science students together similar to the CGT 345. Classes consisted of multiple projects that focused on a different topic or theme. The art students had a hard time working with the editor but were eventually able to adapt while computer science students had an easier time with the class. In contrast, our course covers both art and programming subjects that align with the two disciplines. Also, past history had the art exercises being the most difficult to complete instead of the programming exercises.

Dondlinger and Wilson (2012) created an alternate reality gaming course that focused on a learn and apply approach. The course uses a similar approach to our course in that students are given basic lessons to learn about the application of the technology but the main learning component of the course was solving a problem or applying the solution to a project. One problem with this strategy was that the core learning concepts were not established. The paper blames most of this on the class structure and the late introduction of the game. Our course will need to avoid these problems by introducing the course project at the correct time. However, this course focuses on alternate reality gaming where user's physical actions are more important than the digital interface. Also, they had a smaller class size, six students, participate in the course.

2.3 Summary

This chapter discussed the history of game development courses and how they compare to the current implementation of the introductory game development course at Purdue University.

CHAPTER 3. METHODOLOGY

This chapter discusses the implementation of the game development course and how it was evaluated. The goal of this research is to study student impressions and course performance in order to make suggestions for future courses.

3.1 Participants

Students and instructors that took part of the fall 2014 CGT 345 course at Purdue University were used as the subjects for the study. The course started with 21 students and ended with 19 actively participating in the course. These 19 students were over the age of 18 and were given an optional survey to complete at the end of the course. Surveys were distributed through the course email list and were completed without the instructor present. Instructors kept a log of the course that described issues and observations about the course.

3.2 Course Structure

The course used a lab orientation throughout the semester. Class took place inside a computer lab for two hours every Monday and Wednesday. The beginning of class was used for some lecture and demonstration while the rest of the time allowed students to complete assignments and ask questions. Each week students were assigned an exercise

that explored a different aspect of game development. Students were also given a team project near the middle of the semester to complete.

3.2.1 Lecture

Lectures consisted of demonstrations that would go over that week's exercise. Every Monday a new exercise was introduced by showing students how the completed exercise would function. Depending on the exercise, a demonstration of concepts or components would be shown to the students.

As a demonstration for Exercise 00, students were shown how to navigate through the Unreal editor. This started with showing students how to create a project. Next, a quick overview was done of the different functions of the windows within the editor. This ranged from discussing how to move within the level editor to how certain windows described what was happening within the editor. Students were then given the rest of the class time to experiment with the editor and ask questions.

The demonstration for Exercise 02 highlighted the different tools of the landscape editor. Students were shown how to set up a landscape by creating a material and using particular editor settings. Examples were then shown of the different tools that could be used to change the topography of the landscape.

To combat issues seen in previous iterations of the course, a demonstration was given for Exercise 04 that showed what not to do when modeling a house. Students received an explanation of examples with improper UV's and normal's. They were then shown a correct house and how it differed from the incomplete examples.

Due to the lab oriented style of the course, there were few lectures as the course had an emphasis on students' exploration and asking questions. Students would be introduced to their work but a greater importance was placed on exploring the editor and having one-on-one discussions with the instructor. This gave them time to understand how to create games before discussing how to create a good game.

3.2.2 Exercises

Every week, students were given an exercise to complete. Each exercise built upon the previous one until a completed game was created. These finished projects featured similar objectives with small aesthetic changes.

Figure 3.1 Exercise 00 Start

Exercise 00, Figure 3.1, was an introduction to the Unreal Engine with the goal of teaching students about Unreal's scalability settings and how to package a project. Students opened up the editor and used the Third Person Blueprint Template as the basis for the assignment. The Unreal Editor featured ways to edit scalability settings while in the editor. These settings affect properties such as view distance, shadows, or other render properties. Changing these settings allowed the game to run smoother on slower machines. When a project was turned into an executable format, Unreal calls this packaging, the scalability settings do not transfer over. To fix this, students were given a

guide to add these settings to a built project. The finished exercise was a packaged project with some type of engine scalability applied.

Figure 3.2 Exercise 01 FPS Controller

In the next exercise, shown in Figure 3.2, students created a first person controller. This controller used the WASD keyboard keys and the mouse to move around the level. The controller was also able to jump with the space bar by utilizing Unreal's input mapping and C++. Due to unforeseen lab issues, some students were unable to complete the exercise until a later date. The students that were able to complete this assignment had a home computer that was able to run Unreal Engine. In order to include the other students, a Blueprint version of the assignment was created. Blueprint is a node-based programming tool that comes with the Unreal Engine. It replaced the previous node-based programming tool in Unreal 3 called Kismet. These lab exercises contained the same content as the original but without the C++ programming. Another feature that students experienced was how to use the Game Mode class. This feature sets certain universal values that would affect how the game would be played. The goal of the exercise was to introduce students to the different programming methods within Unreal by also introducing them to the concept of character controllers. In the end, students had

a working controller that would allow them to move and eventually interact with the objects.

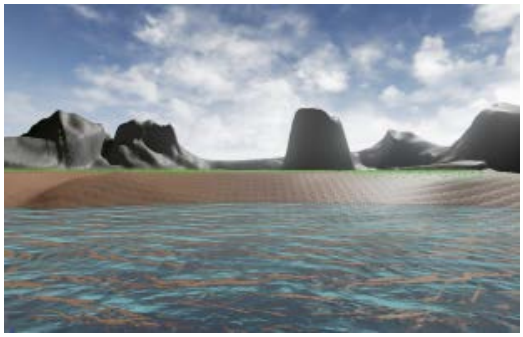


Figure 3.3 Exercise 02 Landscape

Exercise 02 focused on creating a landscape to explore. Students were introduced to the material editor where they would create the layers in order to paint details onto the blank terrain. Next, they used the landscape editor to create a landscape. The editor allowed them to change the height and composition of the landscapes. Grass was also placed using the foliage tool. A grass mesh was provided for the students. Finally they followed a tutorial to create a basic water effect for an ocean. Students were shown how to utilize the material editor and edit the terrain using the landscape tool. The end product was an island, shown in Figure 3.3, that can be explored using the player controller from the previous assignment.

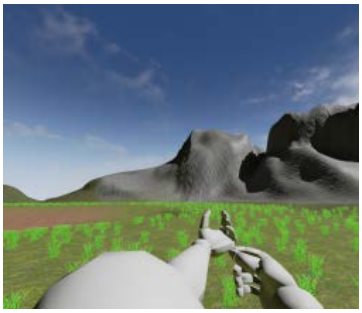


Figure 3.4 Exercise 03 FPS Mesh

Students added a mesh to the character controller in exercise 03. This made it similar to a FPS controller used in AAA games such as Call of Duty except without the aim assist. They added a character mesh that would be invisible to the player camera and an arm mesh that would only be visible to the player camera. Also, students learned about applying and sequencing animations for the meshes. The goal of this exercise was to show students how to utilize meshes with an animation in the engine. With this, the major functions for the character controller were completed.

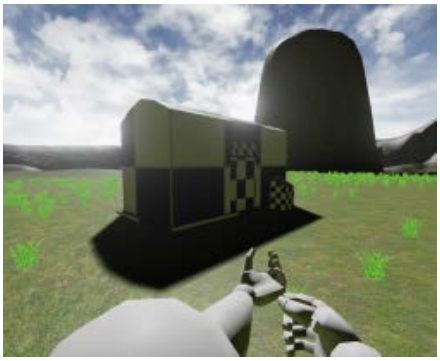


Figure 3.5 Exercise 04 House 1

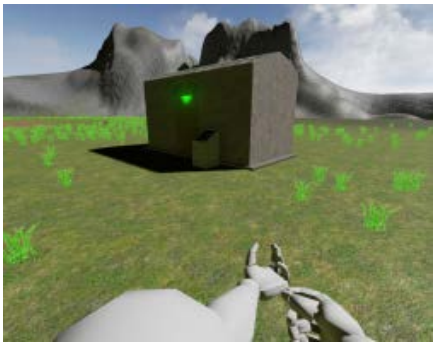


Figure 3.6 Exercise 05 House 2

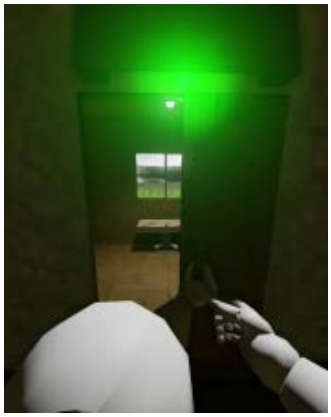


Figure 3.7 Exercise 06 House 3

The next three exercises contained the modeling problems experienced in previous iterations of the course. Exercise 04, Figure 3.5, had students create a house model along with other household objects. By expanding on the CGT 241 work, students learned about how normal's and UV's affect the light maps of models in the Unreal Editor. Next, students made aesthetic changes to the models by adding textures and lights in exercise 05, Figure 3.6. Finally, they added collisions to the house and an open door sequence using Blueprint in exercise 06, Figure 3.7. The end goal for these three exercises was for students to understand how to create, implement, and interact with models within the Unreal Engine. Together, students completed a house door that would open and close depending on the controller's proximity.

Figure 3.8 Exercise 07 Battery

Exercise 07 introduced students to Unreal's HUD (Heads-Up Display) elements. First they created a battery mesh to spawn along with an image representing the different power levels. Students learned how to randomly spawn objects within an area and how to collect them. They then used the HUD Blueprint to inform the player of the number of batteries they had collected. The house Blueprint was changed to only open when the character has collected all of the batteries. This exercise taught students how to create a HUD and communicate between different programming scripts. In the end, the character collected randomly spawning batteries to open the door of the house, Figure 3.8.

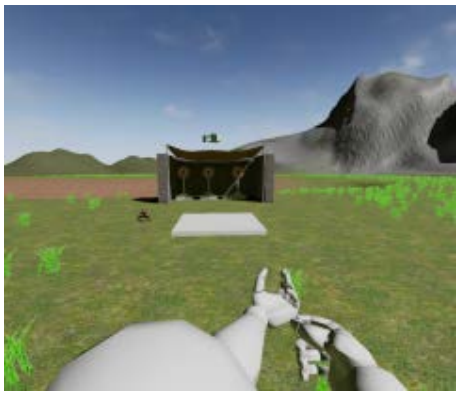


Figure 3.9 Exercise 08 Target Range

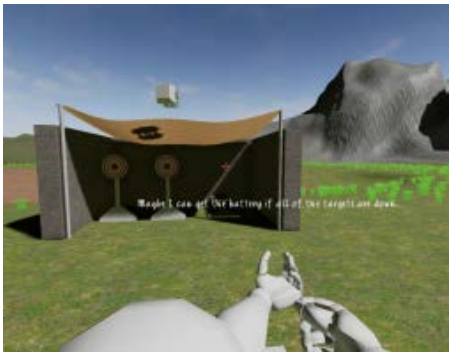


Figure 3.10 Exercise 09 Projectiles

Next, students were tasked with creating a target range, three targets, and a coconut for Exercise 08, Figure 3.9. This included texturing the models of the different

objects. These objects were used in Exercise 09 where the character controller was able to fire a projectile, which was a coconut. They fired the coconut to try and knock down all of the targets in order to gain another battery. These exercises taught students about collisions and how to apply basic game logic. The completed projects have an interactive shack that rewards the player for completing the shooting range, Figure 3.10.

Figure 3.11 Exercise 10 Particles

Exercise 10 has the character collect a match, one of the other models created in Exercise 04, in order to light a fire, Figure 3.11. Students experimented with Unreal's Cascade to create a fire particle effect. There were some problems implementing this exercise. The final student version appeared different then the intended result. This could be due to the changes in the exercises or due to new engine settings. Originally, students were going to learn how to migrate assets between projects. This was too much for the students to handle with all the changes so this part of the exercise was cut. Even with this issue, students learned how to create and edit the particle system. In the end, the player had the ability to activate a particle system that signaled the winning condition of the game.

Figure 3.12 Exercise 11 Main Menu

In Exercise 11, a main menu was added to the project, as shown in Figure 3.12. This exercise expanded on the HUD lesson from Exercise 07 by allowing students to create clickable buttons. The menu allowed the player to play the game, give instructions, and quit the game. Students were taught about some of the shortcomings of the Blueprint editor. One of the problems is that a new line command does not work in Blueprint. This means that it will be difficult to write paragraphs of text for their games. By completing this exercise, students understood how to create a working menu that can take them to the main game.

Figure 3.13 Exercise 12 Polish

The last exercise, 12, added a loading and win screen to the game. The terrain and other assets needed to load when going from the main menu to the game. Without the

load screen, the player would notice certain issues with the level. Adding a loading screen gave the game time to fix the issues and gave the player a smoother experience. This exercise showed students how to create this screen and a win screen after the player lights the fire. The goal of the exercise was to show students the logic of screens within games. All together, the students had a completed project where they transitioned from different levels to complete objectives in order to win the game.

Table 3.1 *Final Exercise Plan*

Title	Description
Exercise 00: Start	Introduction to Unreal
Exercise 01: FPS Controller	Character Control and Inputs
Exercise 02: Landscape	Landscape tool and Material Editor
Exercise: 03: House 1	Modeling and UV unwrapping a house
Exercise 04: House 2	Texturing House
Exercise 05: Target Range	Create a target range
Exercise 06-1: Unity	An introduction to the Unity Engine
Exercise 06-2: Landscape Unity	Create a Terrain/Landscape using Unity
Exercise 07: FPS Mesh	Adding mesh and animation to controller
Exercise 08: House 3	Fire Projectiles to shoot down targets
Exercise 09: Battery	Spawn and collect items
Exercise 10: Particles	Creating fire using cascade
Exercise 11: Projectile	Fire projectiles to shoot down targets
Exercise 12: Main Menu	Creating a menu
Exercise 13: Polish	Adding a Load and Win screen

There were problems using the Unreal Engine with the lab environment. Due to the way that the system stored data and how Unreal read files, certain versions of the project could not run in lab and there were compatibility issues with compiling code in Visual Studio. Exercises had to be moved into an order that could be completed successfully in lab until a solution was found. Table 3.1 illustrates the changes to the

schedule and adds a description of the exercises. Unity exercises were added to the class to try and compensate for the problems.

Exercise 6-1 was an introductory lesson into Unity where the students created a controller that can shoot and knock down objects. The goal was to understand how Unity handles objects and code. On the other hand, exercise 6-2 had students create the same landscape in Unity. This helped to confirm the basics of landscape, or terrain, development in a game engine. Also, students were given a small sample of the Unity workflow to compare to Unreal.

In the end, a partial solution was found where settings had to be saved in the project's config files as well as converting all of the C++ sections into Blueprint for the lab computers. This conversion process created two versions of the exercises, home and lab. The home version of the exercise was close to the original but with a few changes due to the reordering of the exercises. These were created for students that worked mainly on their home computers. In contrast, the lab version was created for students that could only run Unreal in the lab environment. This section implemented the Blueprint solution instead of relying on Visual Studio. Students still used the same process and terminology but the home students got to experience the C++ and visual programming interface while the lab students only experienced the latter.

It was discovered in the next semester that there were some issues with the lab version of Visual Studio. It was not recognizing certain file directories. To fix this, students went to the project properties and went to VC++ Directories to set everything to inherit from parent. This allowed students to compile their code. However, students were still unable to package projects that had opened Visual Studio. The Config folder issue

was also solved after updating the Unreal Engine to 4.4. This update automatically saved the changes done to project settings in contrast to the previous versions that were dependent on the Set as Default button. It was also discovered that Unreal would only work properly within version 4.4 to 4.6. Anything above or below it did not work correctly in lab.

3.2.3 Team Project

Near the middle of the semester, students worked together in team projects to create a game. Teams consisted of 3 to 4 students that decided on the game that they would create. Students were given the freedom to choose their own engine to use for the project. The Professor and TA acted as advisors for the project and set up milestones each week for the student to complete. These milestones were modeled on the publisher and developer relationship seen in the video game industry. The goal of this project was for students to explore the engine and to solve the issues they would face while creating a real game.

3.3 Experimental Design

Students had the option to complete a survey at the end of the semester. This survey was hosted on Qualtrics and distributed through the course email. No names were recorded in the survey and the course email does not reveal the names of the students on the list. Survey completion was not mandatory for the course and the instructor was not present when they completed the survey.

The main purpose of the survey was to gather a mix of quantitative and qualitative data on the performance of the course. It asked students their thoughts on certain components of the course and to explain their thought process. The questions with quantitative results asked the students to evaluate a positive statement with a seven point Likert scale from Strongly Agree to Strongly Disagree. A scale of seven was used as it gave a greater range of responses and it fit well with the survey format. The qualitative results were open-ended and asked why the student answered in that manner. The quantitative data created suggestions for future studies while the qualitative results explained the suggestions. Student grades were also used in the study. These grades were not identified with the student in anyway and were only be used to judge the overall performance of the course.

Instructors kept a log that described different problems and observations seen throughout the semester. These observations were compared with the student's in order to account for possible bias from the suggestions.

3.4 Hypothesis

H_o = the course had no effect teaching students basic game development skills to advance to the next level of game development.

H_a = the course had an effect teaching students basic game development skills to advance to the next level of game development.

3.5 Analysis

The quantitative data were analyzed using a *t*-test and confidence interval using an alpha value of .05. Due to the small sample, any results were seen as a suggestion rather than significance until future studies are done. Qualitative data from the students and instructors were analyzed to find common values to describe the experience of the course.

3.6 Summary

This chapter discussed the methodology by describing the assignments for the course and how the performance will be evaluated. Also, this illustrated the hypothesis and participants for the study. The next chapter will go over the results.

CHAPTER 4. RESULTS

This chapter illustrates the results and discusses the findings. Participants needed to complete every section of the survey except for the last page that contained open-ended questions. Testing was done using confidence intervals due to the small sample. Any statistical analysis was taken as a suggestion for future research.

4.1 Participants

A total of 21 students participated in the course. Of these students, 19 actively participated in the course by completing exercises and team projects. 2 students completed less than 1% of the course work. 17 students completed the survey while 14 completed the requirements for the study.

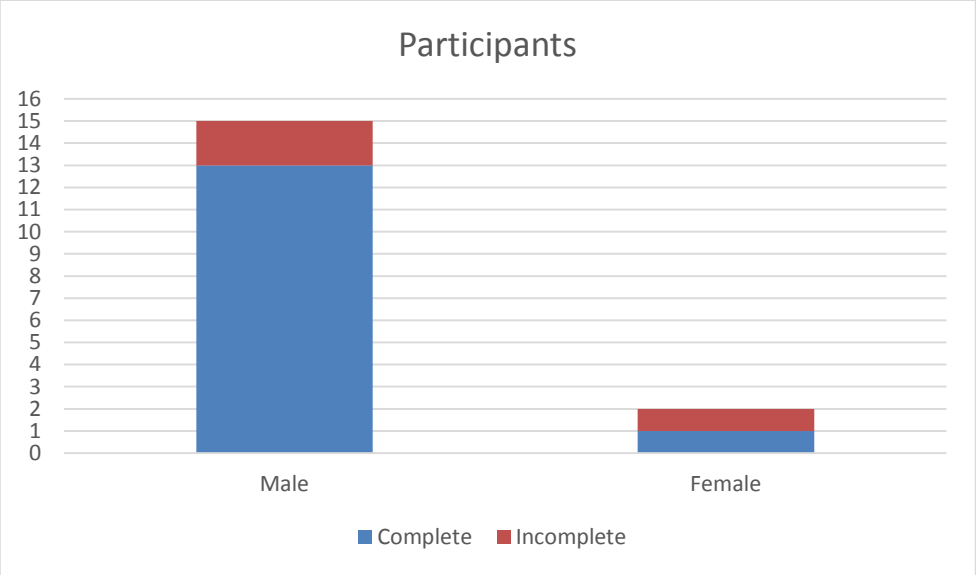


Figure 4.1 Participants' Gender

At the end of the semester, the course had a total of 19 male and 2 female students. Of these students, 15 males and 2 females attempted to complete the survey. Two males and 1 female were unable to meet the completion requirements for the study.

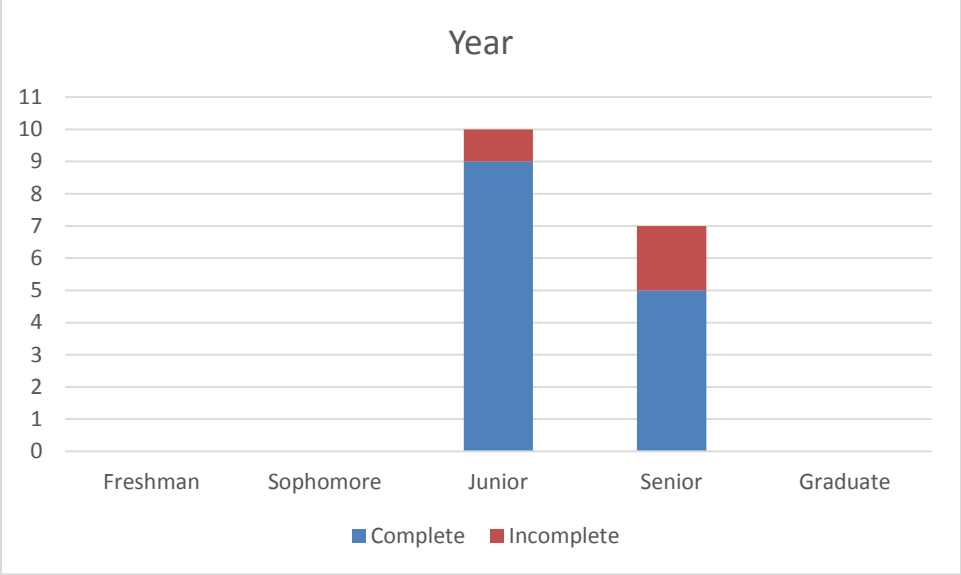


Figure 4.2 Participants Academic Year

The participants of the survey were upperclassman. 10 of the participants were Juniors while 7 were Seniors. 1 of the Juniors and 2 of the Seniors did not complete the survey.

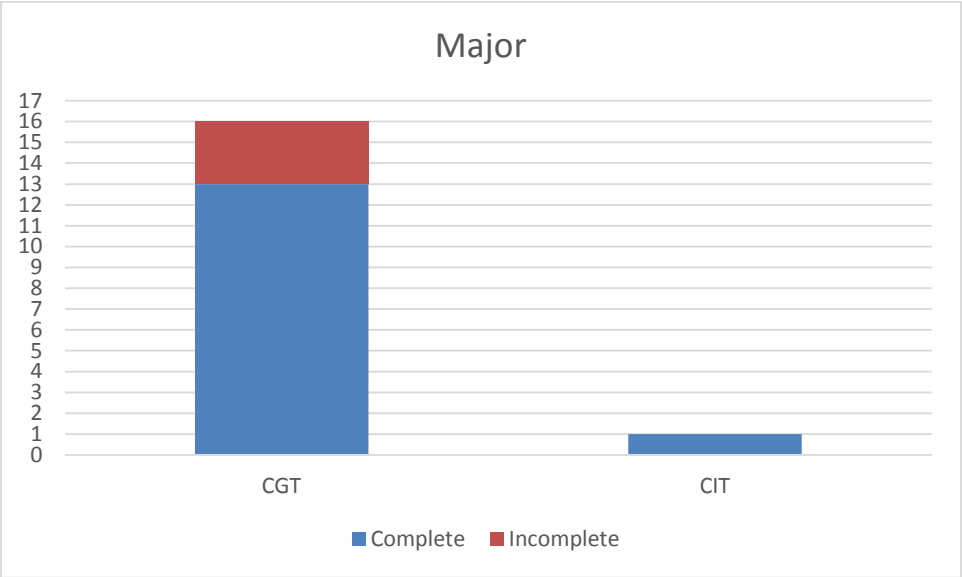


Figure 4.3 Participants Major

16 of the participants were from the CGT department while 1 student was from the Computer Information Technology, CIT, department. The 3 students that did not complete the survey were from the CGT department.

Participants also self-reported modeling, coding, or game software that they were familiar with. A majority of students reported having prior skills with the Autodesk Maya modeling software. Some students also had used other modeling software such as Blender, 3DS Max, or Catia. There was a trend in programming were students had either used some type of C coding or Java. A couple students also had some experience with Unity 3D engine. One student had prior experience using the Unreal Engine before taking the course. Another student reported having no previous skills before taking the course.

4.2 Results

This section shows the results by splitting them into the categories of prior experience, exercise performance, team project performance, course format, and course performance.

4.2.1 Prior Experience

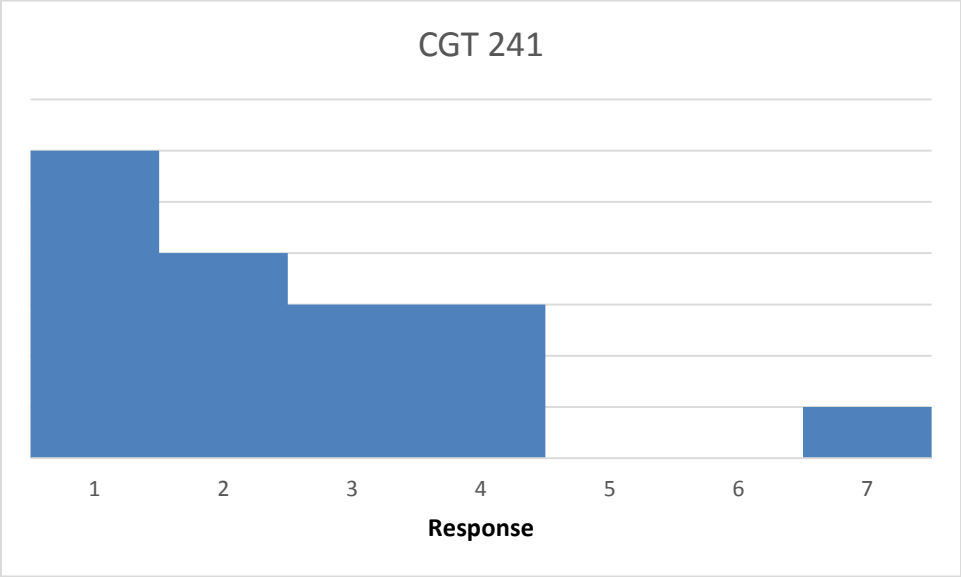


Figure 4.4 CGT 241 Usefulness Chart

Table 4.1 *CGT 241 Usefulness Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	7	2.47	2.64

Table 4.2 *CGT 241 Usefulness Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.62	17	-3.894	1.784, 3.156

When asked about the usefulness of CGT 241 for the course, the data shows a right skew favoring a positive response. Along with the confidence interval of 1.784 and 3.156 not containing the null hypothesis value of 4, this suggests that students felt that CGT 241 was useful in completing the course.

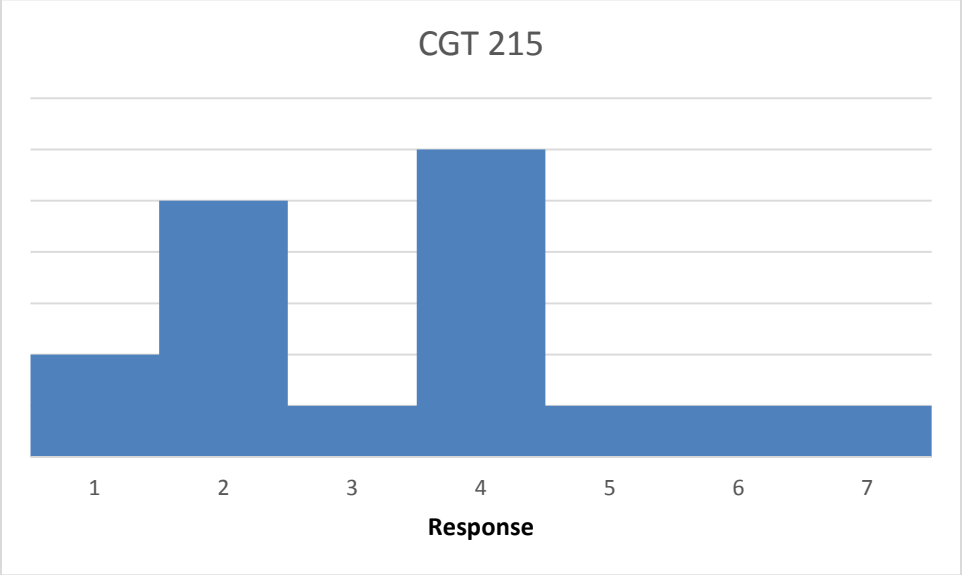


Figure 4.5 CGT 215 Usefulness Chart

Table 4.3 *CGT 215 Usefulness Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	7	3.35	2.87

Table 4.4 *CGT 215 Usefulness Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.69	17	-1.586	2.634, 4.066

Participants were asked about the usefulness of CGT 215 for completing the course. The data for CGT 215 usefulness has a slight right skew. Responses had a wide spread making it closer to a normal distribution. Due to the confidence interval

containing the null hypothesis of 4, 2.634 to 4.066, the data suggests that CGT 215 had no effect in preparing students for the course.

4.2.2 Exercise Performance

Figure 4.6 Exercise Difficulty Chart

Table 4.5 *Exercise Difficulty Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	2	7	4.5	2.58

Table 4.6 *Exercise Difficulty Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.61	14	1.162	3.738, 5.262

The data regarding the difficulty of the course exercises has a slight left skew. No one strongly agreed with the statement. Also, the data has a confidence interval of 3.738

to 5.262 that contains the null hypothesis of 4. This suggests that students had a neutral view on the difficulty of the course exercises.

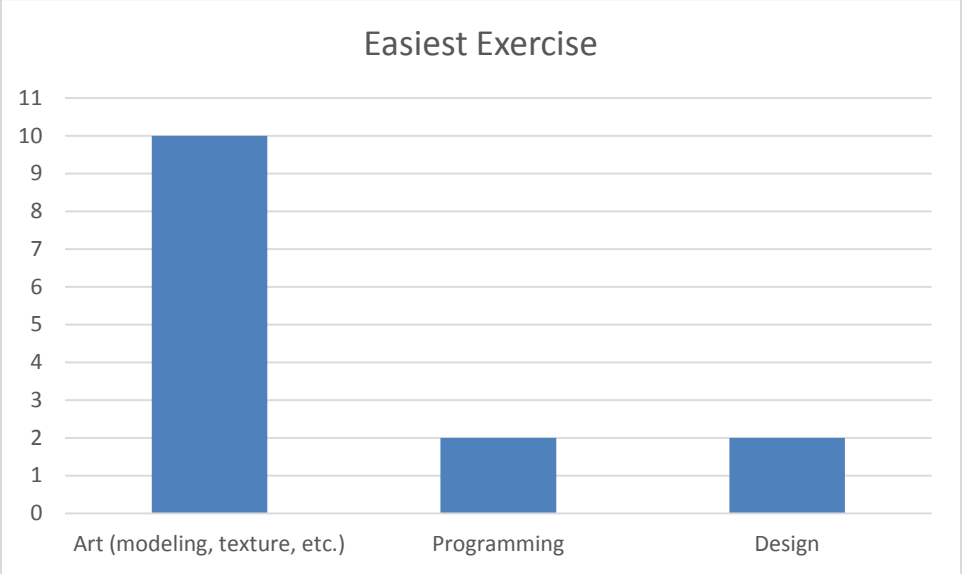


Figure 4.7 Easiest Exercise Chart

Participants felt that the art-oriented exercises were the easiest. They reported that their past modeling experiences helped them to complete the exercises. Another reason that participants felt that it was easier was because Maya was working correctly in labs while Unreal was experiencing some issues with the programming components.

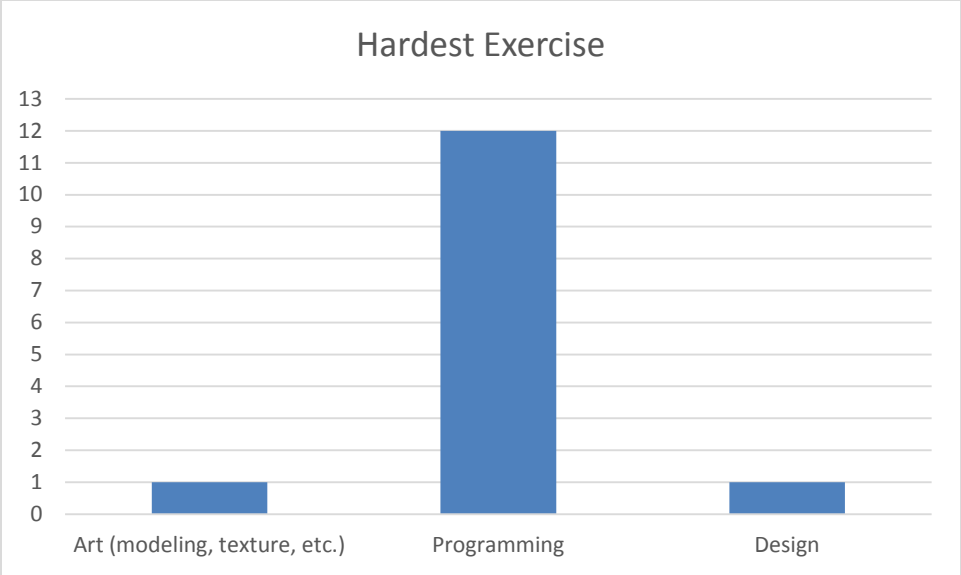


Figure 4.8 Hardest Exercise Chart

Participants felt that the programming exercises were the hardest due to a lack of experience. They reported that they did not understand the Blueprint interface or how to use C++ with the Unreal Engine. Again, some participants talked about the lab issues and how it affected their ability to access code. Participants also felt that CGT 215 did not prepare them for the course. There was too much of a gap between the introductory programming and what they needed to accomplish in the course.

When asked which exercise would be the most useful, most participants felt that the exercises that were oriented towards the art aspect were the most useful. Specifically, participants enjoyed the terrain and house exercises. They reported that the exercises allowed them to understand how Unreal handles modeling. One student preferred the Unity exercises over the Unreal exercises.

In order to improve the exercises, participants wanted a greater amount of images to be added in the exercise instructions. They felt that they could sometimes get lost

within the large amount of text. Also, they would either prefer more demos or more of an explanation of the exercises in class.

4.2.3 Team Project Performance

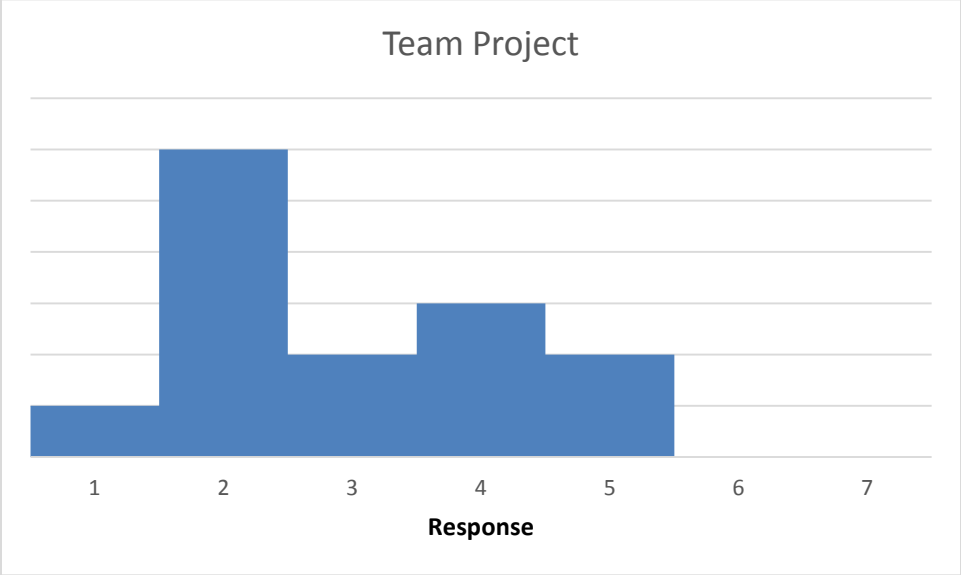


Figure 4.9 Team Project Performance Chart

Table 4.7 *Team Project Performance Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	5	2.93	1.61

Table 4.8 *Team Project Performance Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.27	14	-3.152	2.323, 3.531

Participants' data regarding the team project performance has a right skew. It had a maximum value of 5 – somewhat disagree. The confidence interval was also 2.323 to

3.531 which does not contain the null hypothesis of 4. This suggests that team projects went well during the course.

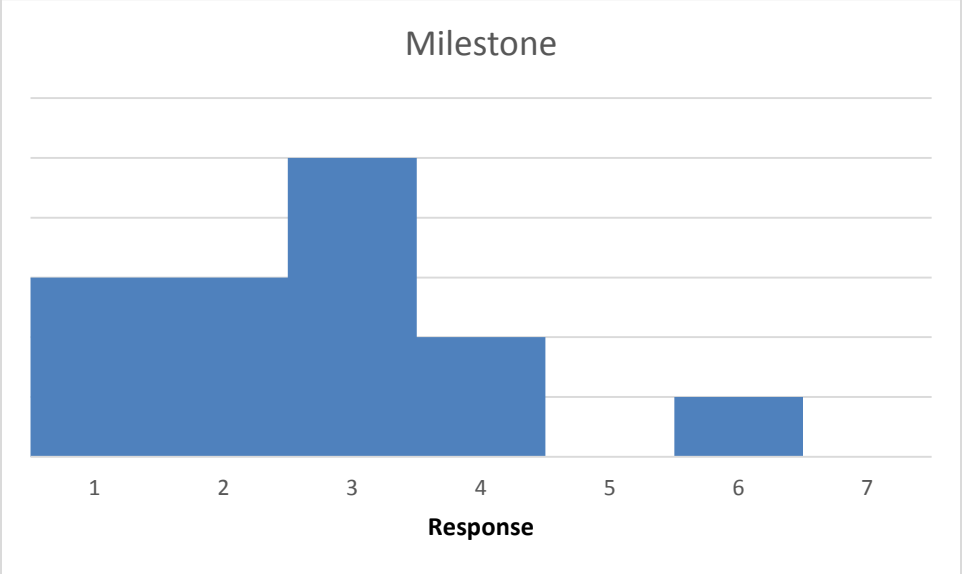


Figure 4.10 Milestone Usefulness Chart

Table 4.9 *Milestone Usefulness Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	6	2.71	1.91

Table 4.10 *Milestone Usefulness Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.38	14	-3.498	2.057, 3.363

When asked about the team project milestone usefulness, the results show a right skew. The maximum value was a 6 for disagree. This along with the fact that the confidence interval, 2.057 to 3.363, did not contain the null hypothesis, 4, suggests that the milestones were helpful.

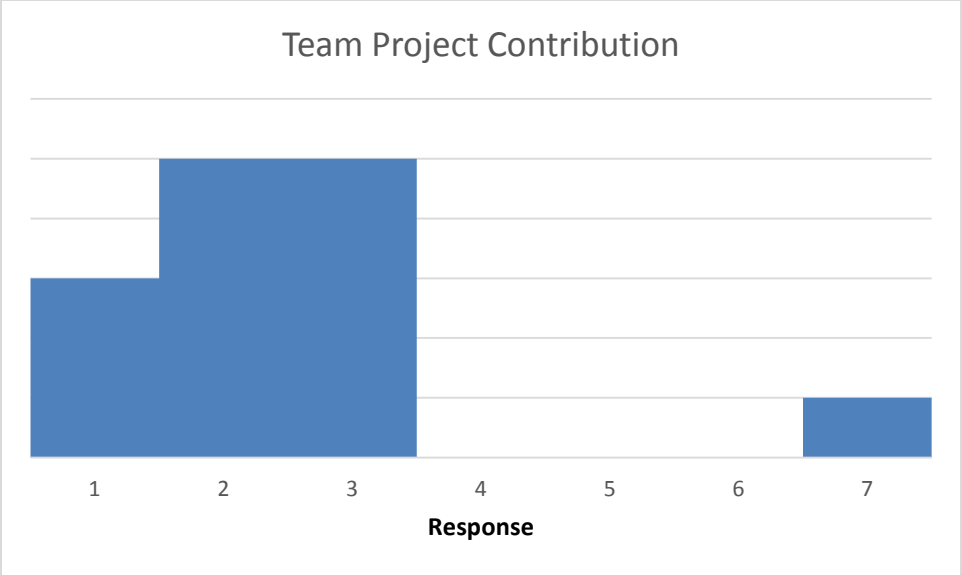


Figure 4.11 Team Project Contribution Chart

Table 4.11 *Team Project Contribution Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	7	2.5	2.27

Table 4.12 *Team Project Contribution Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.51	14	-3.717	1.785, 3.215

The data for team project contribution has a right skew. The maximum value, 7, can also be considered an outlier due to its distance from the main portion of the data. This will not be taken out though as it describes the issues one group faced while trying to complete the project. With a confidence interval of 1.785 to 3.215, the data suggests that the team members had an equal contribution to the team projects.

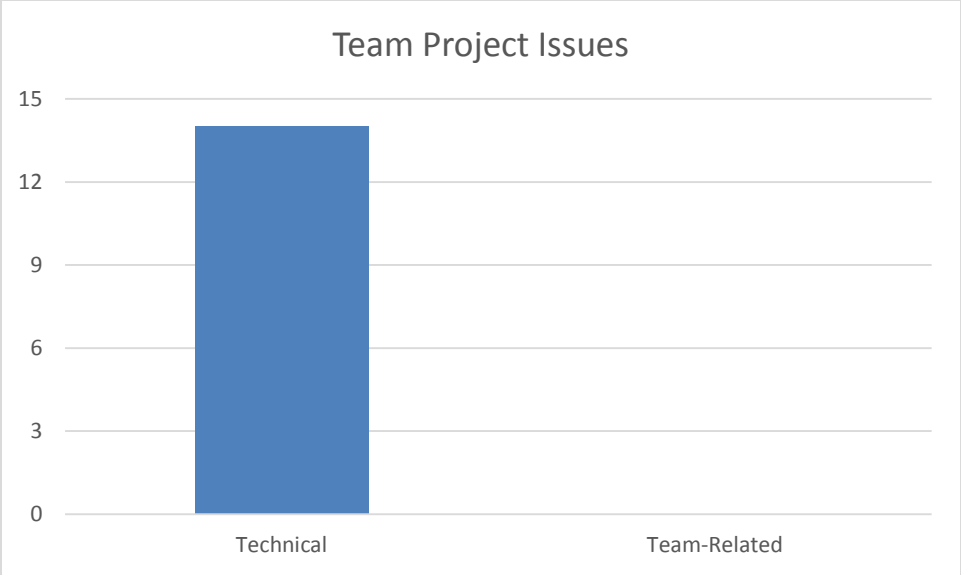


Figure 4.12 Team Project Issues Chart

All participants reported having technical issues with their team projects. Participants reported having a hard time getting used to the engine and figuring out how to implement their game ideas. Once again, the lab problems with the Unreal Engine were brought up as issues.

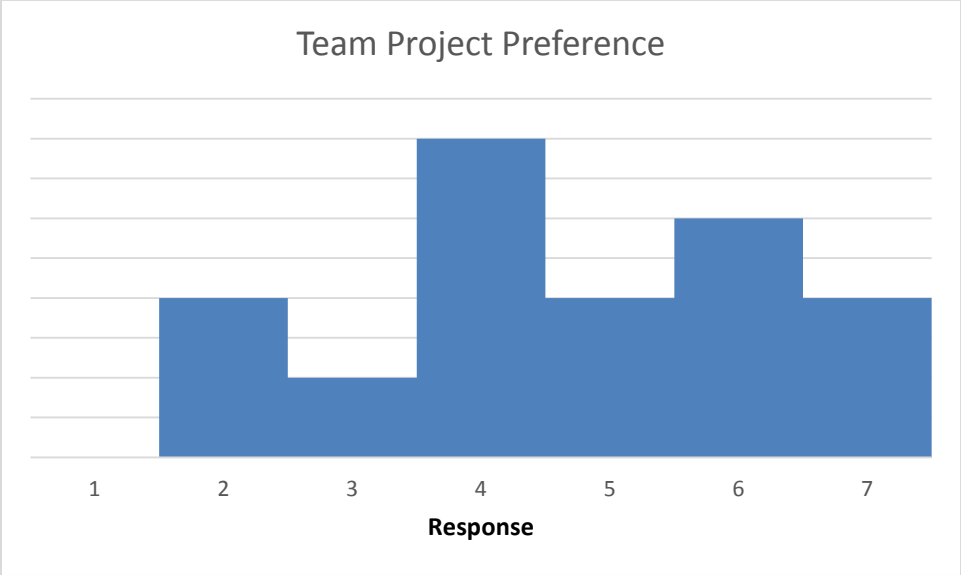


Figure 4.13 Team Project Preference Chart

Table 4.13 *Team Project Preference Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	2	7	4.64	2.71

Table 4.14 *Team Project Preference Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.65	14	1.451	3.859, 5.421

Participants were asked if they would prefer a large individual project over the current team project. The data showed a slight left skew and no one strongly agreed with the idea of having large individual projects. The confidence interval, 3.859 to 5.421, contains the null hypothesis of 4. This suggests that the participants had no preference for the type of project given to them.

Figure 4.14 Exercise and Team Project Chart

Table 4.15 *Exercise and Team Project Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	7	4.07	4.38

Table 4.16 *Exercise and Team Project Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	2.09	14	.125	3.081, 5.059

While the largest portion of participants somewhat agreed that the exercises helped the project, participants also had a large number that chose “disagree” and “strongly disagree”. Some students stated that the exercises helped them to understand the engine while others said that the exercises did not help because their game covered a different genre. The confidence interval, 3.081 to 5.059, contained the null hypothesis, which suggests that participants thought that the exercises did not help to complete the team projects.

Participants wanted to form teams at an earlier time. This would allow them more time to complete the project and understand their teammate’s skills. They would also like the students to split by their skills. By splitting them, the teams could have an even distribution of artists, programmers, and designers.

4.2.4 Course Format

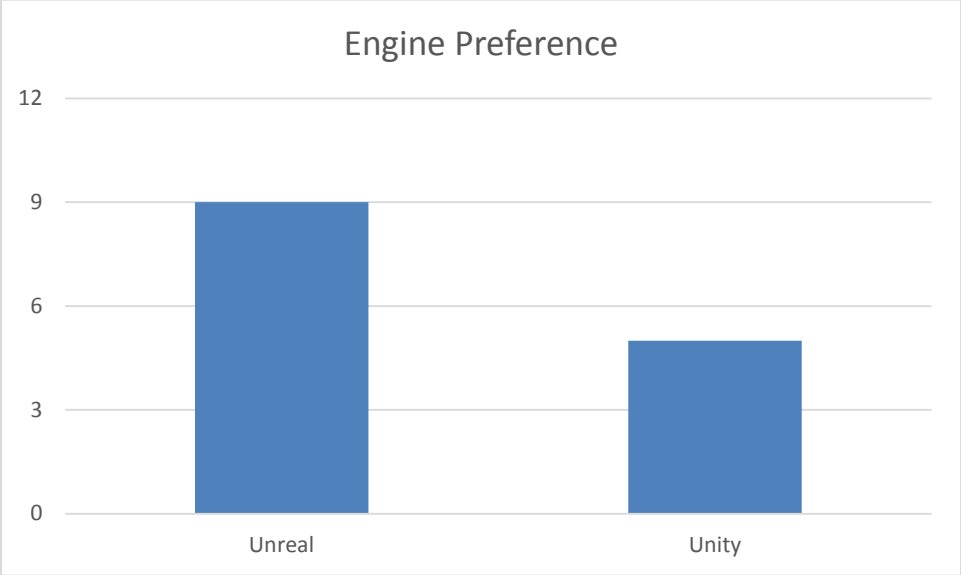


Figure 4.15 Engine Preference Chart

A majority of participants preferred the Unreal Engine over the Unity 3D engine. They reported that the Unreal Engine was capable of doing some amazing things and that it would look good on a resume. Participants that preferred Unity liked it for its simple interface and familiarity.

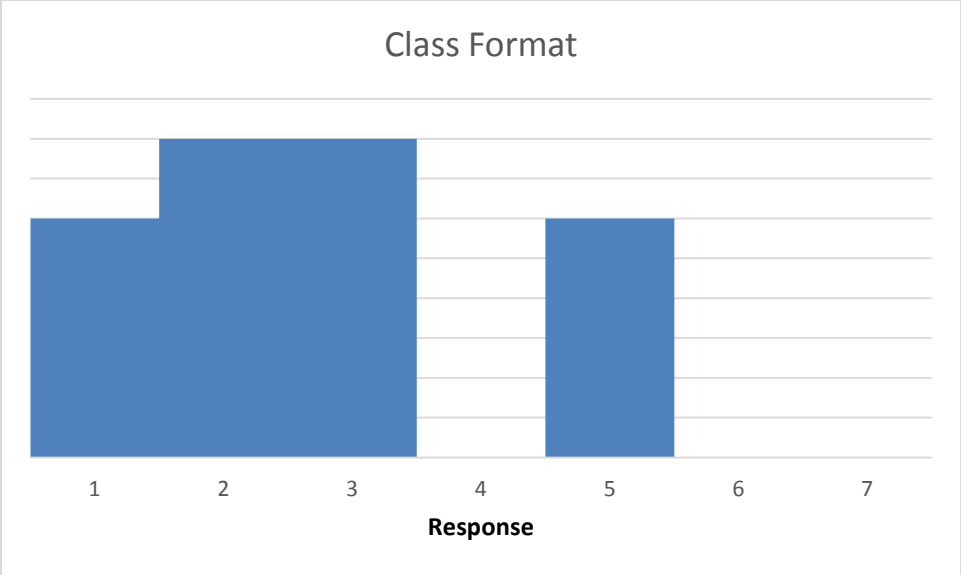


Figure 4.16 Class Format Chart

Table 4.17 *Class Format Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	5	2.71	2.07

Table 4.18 *Class Format Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.44	14	-3.352	2.028, 3.392

When asked if the participants preferred the lab-oriented class, the data had a right skew. The minimum value was 5 for somewhat disagree. Along with the confidence interval, 2.028 to 3.392, not containing the null hypothesis, the data suggests that participants preferred a lab-oriented class over a traditional lecture course.

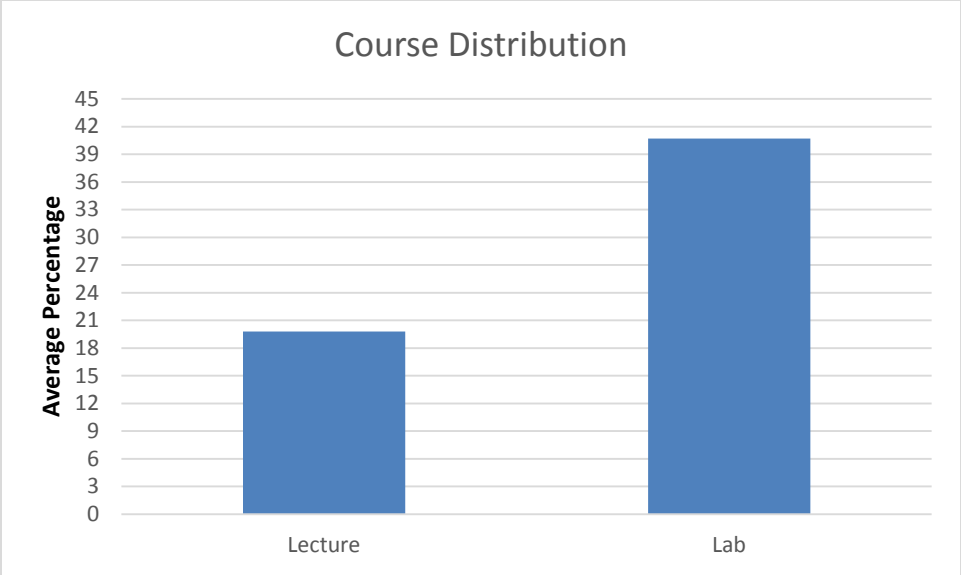


Figure 4.17 Course Distribution Chart

Participants were asked what should be the ideal course format. On average students wanted lecture to take up 19.79% of the course while the lab format on average would take up 40.71%. Along with the results above, the data suggests that participants would like the course to have a larger lab orientation than lecture. If a lecture were to be added to the course, participants wanted it to focus on game development concepts such as coding examples, game design discussions, and discussing the current state of the video game industry.

4.2.5 Course Performance

Figure 4.18 Game Development Confidence Chart

Table 4.19 *Game Development Confidence Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	2	7	3.57	2.42

Table 4.20 *Game Development Confidence Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.55	14	-1.038	2.836, 4.304

When asked about their confidence in game development, the data has a right skew with possible outliers in the maximum value, 7. A majority of students chose "somewhat agree" while 2 students strongly disagreed with the statement. These two answers widened the confidence interval to 2.836 to 4.304. While the confidence contains the null hypothesis, there seemed to be a trend to agree that participants were confident in

their game development skills. However, the data suggests that the students did not agree or disagree regarding how confident they are in game development.

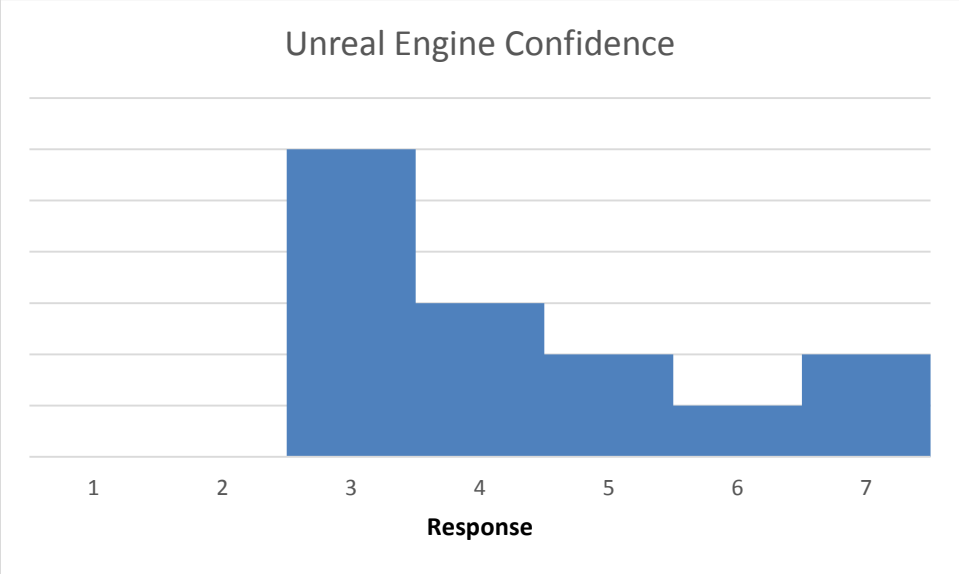


Figure 4.19 Unreal Engine Confidence Chart

Table 4.21 *Unreal Engine Confidence Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	3	7	4.29	2.22

Table 4.22 *Unreal Engine Confidence Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.49	14	.728	3.585, 4.995

While the data for the participants confidence with the Unreal Engine has a right skew, the minimum value is 3 for “somewhat agree”. This leads to a confidence interval of 3.585 to 4.995 that contains the null hypothesis. This suggests that participants were neither confident nor unconfident with their emerging Unreal Engine skills.

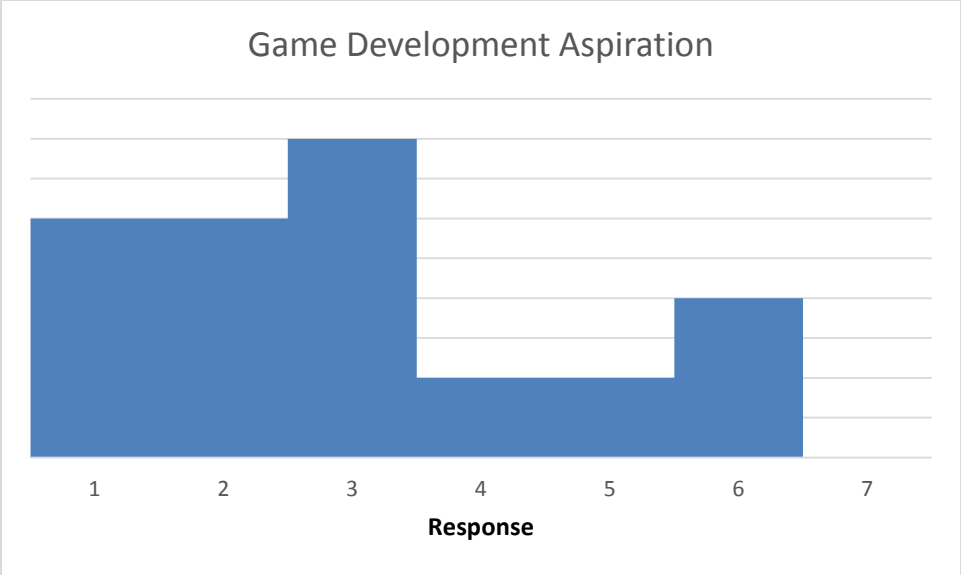


Figure 4.20 Game Development Aspiration Chart

Table 4.23 *Game Development Aspiration Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	6	3	2.92

Table 4.24 *Game Development Aspiration Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.71	14	-2.188	2.191, 3.809

The data has a slight right skew for their aspirations to continue in game development. The maximum value was 6 for disagree. With a confidence interval of 2.191 to 3.809, the data suggests the participants were motivated to pursue game development.

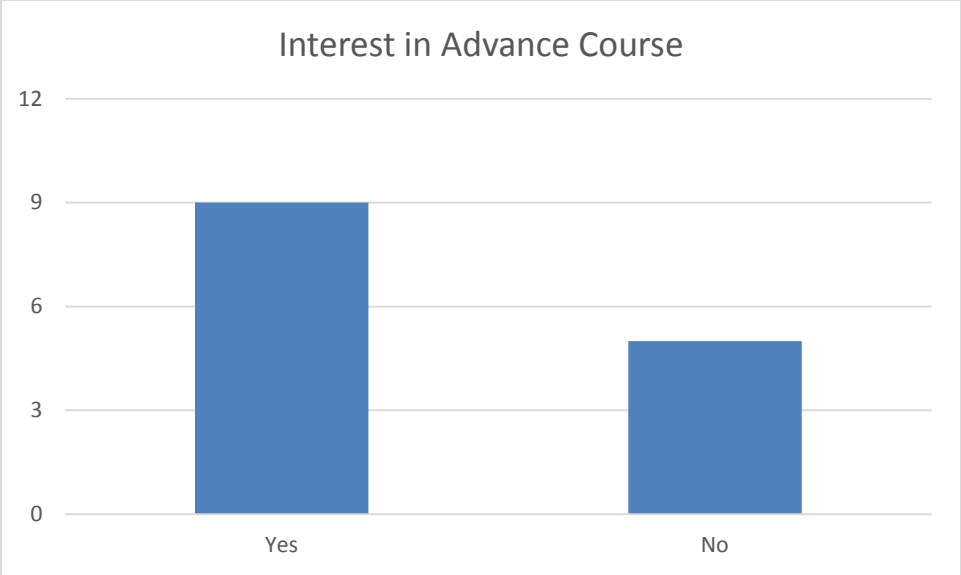


Figure 4.21 Interest in Advance Course Chart

9 of the 14 participants want to continue to the advance game development course. 5 participants do not want to continue in to the next game development course.

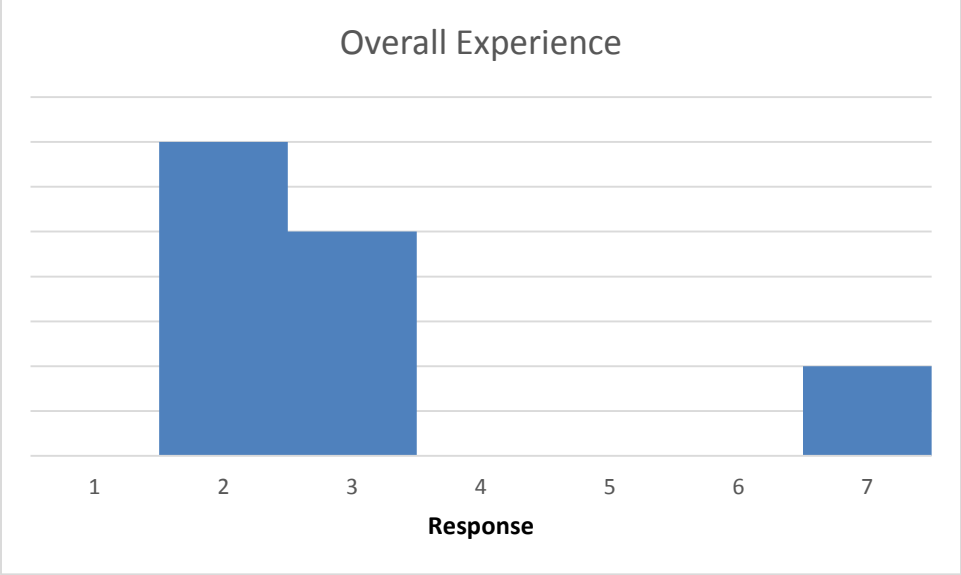


Figure 4.22 Overall Experience Chart

Table 4.25 *Overall Experience Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	2	7	3.07	2.99

Table 4.26 *Overall Experience Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.73	14	-2.011	2.251, 3.889

The data is skewed to the right with a majority of responses ranging in 2 and 3, “agree” and “somewhat agree”. Participants enjoyed the course as they were able to learn how to create games for future career prospects. While some participants reported the engine problems being too much of an issue, most students did not report it as an issue. The confidence interval, 2.251 to 3.889, does not contain the null hypothesis. This suggests that the overall experience of the course was good.

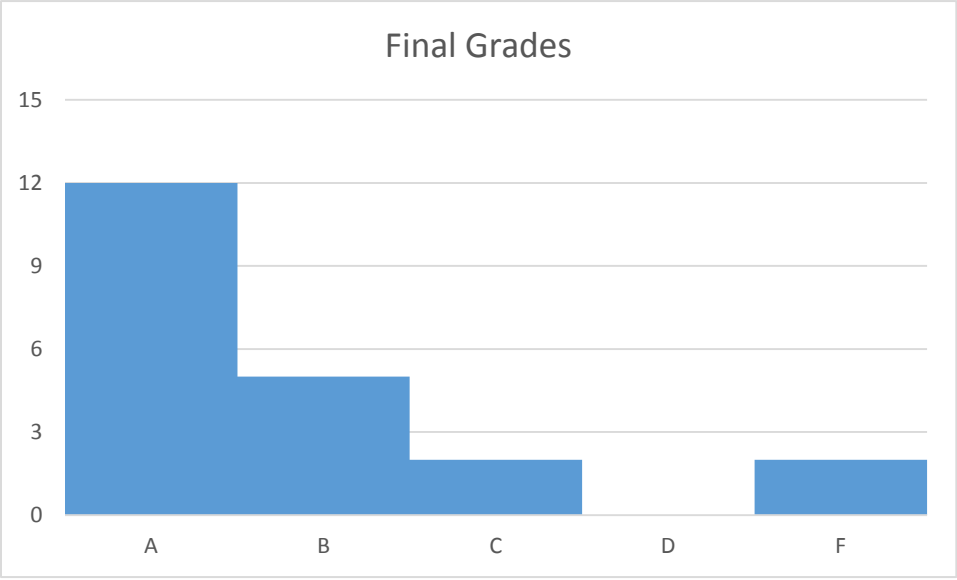


Figure 4.23 Student Grades Chart

Student grades had a right skew with a majority of students receiving an A in the course. These students completed most of the exercises and created a satisfactory team

project. Students that received a B or C did not complete all of the exercises. The 2 F students did not complete the exercises or participate in the team projects.

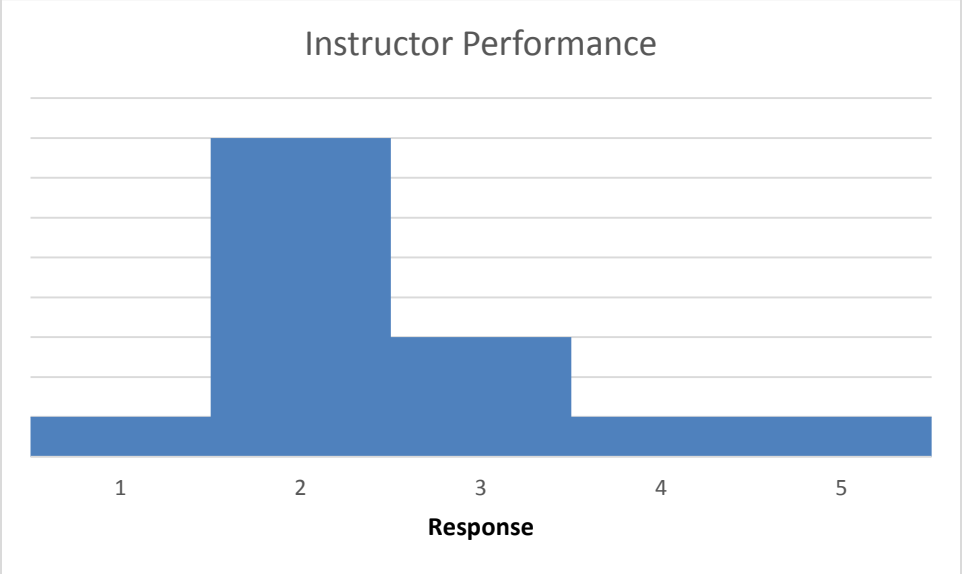


Figure 4.24 Instructor Performance Chart

Table 4.27 *Instructor Performance Data Part 1*

Statistic	Min Value	Max Value	Mean	Variance
Value	1	5	2.5	1.04

Table 4.28 *Instructor Performance Data Part 2*

Statistic	Standard Deviation	Total Responses	t-value	Confidence Interval
Value	1.02	14	-1.834	2.017, 2.983

Participants' report of how well the course was taught showed a right skew along with a confidence interval of 2.017 to 2.983. For this question alone, the null hypothesis was 3 and the Likert scale was set to 5. This difference from the other questions was an unintentional error. Even with this change, the null hypothesis was not within the confidence interval, which suggests that the instructor performance was adequate.

In future iterations of the course, participants would like the technical issues to be fixed along with a greater emphasis on demoing the exercises. Two students reported in the free response that they were disappointed in the course and how it was run. Some students were also disappointed that the course used Unreal instead of Unity. Other than these two instances, the overall results for the course were positive.

4.3 Discussion

The students were responsive with the survey. One of the initial worries for the thesis was that the students would ignore the survey due to it being optional. It is alarming that a large portion of the sample consisted of males but the course only had a few female students. Students were most likely upper classman because the prerequisites for the course tend to be taken during the sophomore year. This course is currently marketed mostly for CGT students. While the course can benefit by introducing it to a wider audience, this audience would need to have the foundation provided by the prerequisite courses.

The problems with CGT 241 seem to have been addressed. Students had minor issues with modeling or creating art assets. There were a few students that needed helped but they were either taking the course at the same time or they had gotten permission to take the course instead of completing the prerequisites. On the other hand, the course revealed a gap with the students' programming knowledge. Students did not understand how CGT 215 helped them to understand programming in the Unreal Engine. This is a gap that will need to be addressed in future work.

While the data in results shows a neutral view on exercise difficulty, there was a trend of students having a hard time completing exercises. Whenever students had an issue with the engine while completing exercises, they generally stopped working to ask for help rather than finding the solution on their own. This is an introductory course but students will need to take some initiative with solving issues when they work in the video game industry. However, the issues in the lab were at the time unexplainable and hindered some of their progress. Another issue some students faced was being unable to understand some of the instructions for the exercises. Students were using different versions of Unreal and things often change between versions. This along with being new to game development made it difficult for some students to complete exercises. Adding additional images to the instructions of the exercises and further exploring their goals should help to improve the students understanding of basic game development skills.

Team projects generally went well. While students had trouble creating content that they were designing, they were able to work together to create some interesting projects. One team experienced some issues when a member stopped showing up for class. This team was then at a disadvantage with one less person working on the project. Thankfully there were still able to work through the problems and to turn in their team project. This group most likely contributed to some of the negative team project results. Team projects could be started at an early time before the mid semester. This would give students more time to complete their projects and to address the technical issues that all of the participants in the study faced. Some students would have liked the team projects to be more structured. This structure would include factors such as dividing skills, setting an exact game project, and choosing a specific engine. While these factors could benefit

the higher course, they go against the exploration mindset we have for these projects. The goal was for students to learn and explore on their own how games are created. The course allows for the student to make mistakes as long as they grow outside of the foundation that is provided. Future course focus on creating a good game but the goal of this course is to learn how to create a game. Participants also reported a neutral stance on the exercises helping the team projects. This could be explained by students choosing projects of a different genre than the exercises and from others that chose to use a different engine than Unreal. The goal of the exercises is to give them the basic idea of how the engine works. They are not there to complete the team project for them. Students need to be able to freely explore in order to understand the difficulties of game development. If restrictions were to be placed, it would be at a lower level course.

A majority of students preferred Unreal over Unity. They understood that it was a powerful engine that could help them find a job. Even if we switched to Unity, students need to be prepared to learn new software when working in the industry. Companies do not always use the same engine, so they will need to be able to adapt and transfer their skills to the new tool. Students also seemed to enjoy the lab format. As an instructor, it was easy to talk to students and help them with their work during class time. However, I was not able to provide more demonstrations or an explanation. The changes to the exercises took a large portion of my preparation time that would have otherwise been spent creating demonstrations. Now that the engine works in lab, it is possible for future iterations to have more demonstrations. Students also reported wanting a larger portion of the course devoted to lecture. While it is good that they want to learn more about game development, this takes time away from teaching them how to make good games.

Discussions on good design are handled at the advance level where students are tasked with creating a fun game.

Students generally did not report positive results for confidence in game development. The question was most likely worded incorrectly as it should have asked if they had learned something instead of an increase in confidence. Students might have actually learned that they do not know enough about game development. This can cause a decrease in confidence even if they receive more knowledge from the course. Also, it was nice to see that a majority of students wanted to continue in game development. This shows that the course is successful in gaining student interest. There were some negative results with students no longer interested in game development. It seems these students were expecting a different course that taught different skills. Overall, students did well in the course if they completed the exercises and did well on the team project. There were some students that did not turn all of their exercises or they showed a bad team project. Either way, students generally passed the course. Even with the issues, students generally enjoyed the course and learned some of the basics of game development.

4.4 Recommendations

In order to prepare a course, there needs to be an understanding of what is taught within and outside of the course. To fix the introductory course, there needs to be some changes made to the overall curriculum.

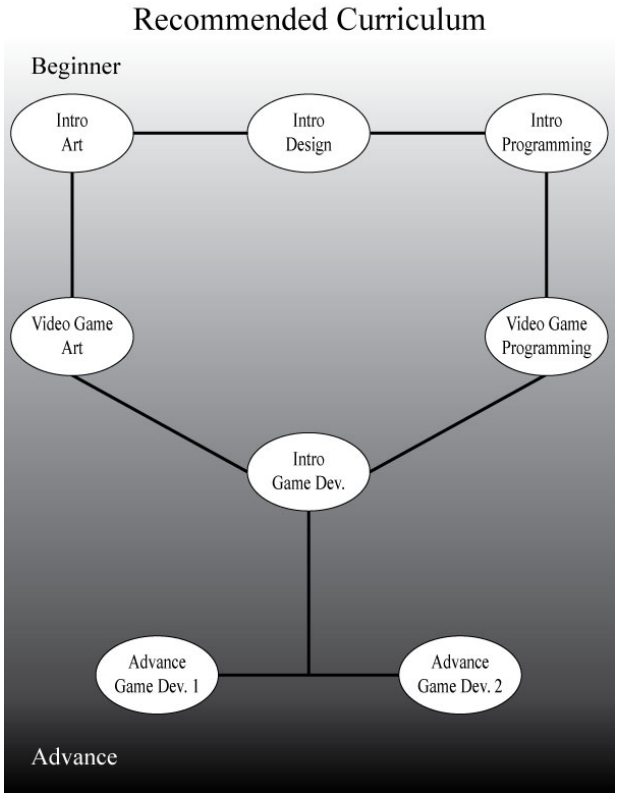


Figure 4.25 Recommended Curriculum

Figure 4.25 depicts a recommendation for the core curriculum for undergraduate game development at Purdue University. It divides the different areas of game development into tracks that expand on certain topics in order to prepare students for a career in game development.

At the beginner level, students need to understand the basics of art, programming, and design. While they may not be going into the se specific fields, they need to have a basic understanding of what is happening in those other fields and how they can work together to create a product. Overall, the goal of the beginner level is to give a basic foundation to students with little to no knowledge in game development and to also draw people into the curriculum. The introductory art course would be CGT 241. According to

the survey and the student's performance, the course is preparing them for game development. The course itself is also going through some changes. At the time that this thesis was written, CGT 241 has just opened up to the entire university in order to draw in more animation students to the department. The introductory game course would be new to Purdue University. Students seem eager to discuss the game industry even before understanding how to create games. This course would be used to give students a basic understanding of the design of games while also getting students interested in game development. Whitehead's course at the University of Santa Cruz is a good example of what this design course would accomplish. Students would play games and discuss why a game is fun or good while also coming up with their own ideas. Introductory programming would be CGT 215. While the course has been suggested to be inadequate in preparing students for game programming, it does teach students the basics of general programming. Game programming needs another course in order to prepare students for the introductory game development course.

At the novice level, students can decide to specialize in either art or programming. This specialization can help them learn about their topic of interest before working on a team. The video game art course would expand on the basic concepts taught in CGT 241 by applying them to a game. The course would be similar to Wynters (2007) in that students would be interacting with an engine but only changing it by applying different models, textures, and terrains. This will decrease the modeling load in the introductory course in order to explore other subjects. The video game programming course was suggested by the data in this thesis. Students are not learning enough about programming within a game development environment. This course would take students through

programming exercises within a game development engine. They would learn how to manipulate objects and to change, store, or access data. This would be similar to DePaul Universities game development course where they use XNA to teach basic coding principles (Linhoff & Settle, 2008). By adding this course, the introductory game development course can continue using its current programming methods.

The intermediate level would be CGT 345, Introduction to Game Development. The focus of this course would be to bring the art and programming students together to create a game. The goal would be the same in that it teaches students how to create a functional game. Due to this focus, students are able to experiment instead of striving to create a "fun" experience. They can try different methods of creating their game idea without worrying if the idea is necessarily fun. Instead, they are learning about the time and effort it takes to create a game.

The advance level would take place over a year, two semesters. Students would pair off into groups to work on a game concept for the entirety of the course. This would take a studio approach and emulate the video game industry. The goal is to create a fun experience that is capable of being published in the market. Students would not only receive experience working in an industry environment but also get a portfolio piece that can help them get a job out of college.

Together, these courses form a curriculum to teach students about game development. It can be expanded as well by adding other options. Electives could be added to expand a student's knowledge. For example, a student can add the video game narrative course as an elective to better understand how stories are written in games. This

recommendation is just for the core curriculum in order to refine the introductory game development course.

4.5 Future Work

While this recommendation is based on data from this thesis and other game development courses, it is still a theoretical curriculum. Further testing needs to be done in order to see if there are any issues and if it is a practical application. Each course has to provide a step into a game development career. This curriculum may not also work for every institution. Purdue University teaches using a general approach that may not work for an art or programming specific background. Either way, those programs need to emulate an industry setting in some form. By testing and reforming the game development program, universities can be closer to having an impact on the video game industry.

4.6 Conclusion

The course had an overall good performance in that it provided students with the basic knowledge to create games. While students wanted to explore areas such as design, it was not the primary focus of the course. A gap was shown in students' knowledge of programming. They felt that the course was too big of a jump from the prerequisite course. This could be solved by expanding the game curriculum at Purdue University to specialize in certain areas. Now that the Unreal Engine is also fixed, it will be easier for students to have access to all of the Unreal Engine's features. A future analysis needs to

be done to see if any other factors influenced the course and to see if the recommendations for the course are correct for the university.

LIST OF REFERENCES

LIST OF REFERENCES

Anderson, E., & McLoughlin, L. (2007). Critters in the classroom: a 3D computer-game-like tool for teaching programming to computer animation students. *ACM SIGGRAPH 2007 Educators Program*. Retrieved from http://dl.acm.org/citation.cfm?id=1282048

Bayliss, J. (2009). Using games in introductory courses: tips from the trenches. *ACM SIGCSE Bulletin*, 337–341. Retrieved from http://dl.acm.org/citation.cfm?id=1508989

Dondlinger, M. J., & Wilson, D. a. (2012). Creating an alternate reality: Critical, creative, and empathic thinking generated in the Global Village Playground capstone experience. *Thinking Skills and Creativity*, *7*(3), 153–164. doi:10.1016/j.tsc.2012.02.001

El-Nasr, M. S., & Smith, B. K. (2006). Learning through game modding. *Computers in Entertainment*, *4*(1), 7. doi:10.1145/1111293.1111301

Gestwicki, P., Sun, F., & Dean, B. (2008). Teaching game design and game programming through interdisciplinary courses. *Journal of Computing Sciences in …*, *8668*, 110–115. Retrieved from http://dl.acm.org/citation.cfm?id=1409791

Graff, E. De, & Kolmos, A. (2007). *Management of change: implementation of problem-based and project-based learning in engineering* (pp. 1–43). Retrieved from http://forskningsbasen.deff.dk/Share.external?sp=S251862f0-a619-11db-b8eb-000ea68e967b&sp=Saau

Graft, K. (2014, July 22). Game Developer Salary Survey 2014: The results are in! Retrieved February 10, 2015, from http://gamasutra.com/view/news/221533/Game_Developer_Salary_Survey_2014_The_results_are_in.php

Hwang, G.-J., Hung, C.-M., & Chen, N.-S. (2013). Improving learning achievements, motivations and problem-solving skills through a peer assessment-based game development approach. *Educational Technology Research and Development*, *62*(2), 129–145. doi:10.1007/s11423-013-9320-7

Linhoff, J., & Settle, A. (2008). Teaching game programming using XNA. *ACM SIGCSE Bulletin*, *40*(3), 250. doi:10.1145/1597849.1384338

McGill, M. (2008). Critical skills for game developers: an analysis of skills sought by industry. In *Proceedings of the 2008 Conference on Future Play: …* (pp. 89–96). Retrieved from http://dl.acm.org/citation.cfm?id=1497000

Owen, D. (2013, April 3). Is it Worth Doing a Degree in Video Games? Retrieved March 15, 2015, from http://www.ign.com/articles/2013/04/02/is-it-worth-doing-a-degree-in-video-games

Pease, M. a., & Kuhn, D. (2011). Experimental analysis of the effective components of problem-based learning. *Science Education*, *95*(1), 57–86. doi:10.1002/sce.20412

Schultz, W. (n.d.). AAA Game. Retrieved April 3, 2015, from http://gameindustry.about.com/od/glossary/g/Aaa-Game.htm

Sung, K., Rosenberg, R., Panitz, M., & Anderson, R. (2008). Assessing game-themed programming assignments for CS1/2 courses. *Proceedings of the 3rd International Conference on Game Development in Computer Science Education - GDCSE '08*, 51–55. doi:10.1145/1463673.1463684

Slick, J. (n.d.). Creating a UV Layout. Retrieved April 3, 2015, from http://3d.about.com/od/3d-101-The-Basics/a/Surfacing-101-Creating-A-UV-Layout.htm

Slick, J. (n.d.). Preparing a Model for 3D Printing. Retrieved April 3, 2015, from http://3d.about.com/od/Creating-3D-The-CG-Pipeline/ss/Preparing-A-Model-For-3d-Printing-Model-To-3d-Print-In-5-Steps_5.htm

Timcenko, O., & Stojic, R. (2012). On Problem Based Learning and Application to Computer Games Design Teaching. *International Journal of Emerging Technologies in …*, *7*(1). Retrieved from http://www.editlib.org/p/44963/

Unreal Engine 3. (n.d.). Retrieved March 16, 2015, from http://www.giantbomb.com/unreal-engine-3/3015-86/games/

Ward, J. (2008, April 29). What is a Game Engine? - GameCareerGuide.com. Retrieved February 17, 2015, from http://www.gamecareerguide.com/features/529/what_is_a_game_.php

Whitehead, J. (2008). Introduction to game design in the large classroom. *Proceedings of the 3rd International Conference on Game Development in Computer Science Education - GDCSE '08*, 61–65. doi:10.1145/1463673.1463686

Wichrowski, M. (2013). Teaching augmented reality in practice: tools, workshops and students' projects. *Proceedings of the International Conference on* …. Retrieved from http://dl.acm.org/citation.cfm?id=2500362

Wood, D. F. (2008). Problem based learning. *BMJ (Clinical Research Ed.)*, *336*(7651), 971. doi:10.1136/bmj.39546.716053.80

Wynters, E. (2007). 3D video games: no programming required. *Journal of Computing Sciences in Colleges*, 105–111. Retrieved from http://dl.acm.org/citation.cfm?id=1181875

APPENDICES

Appendix A IRB Approval Letter

PURDUE
UNIVERSITY

HUMAN RESEARCH PROTECTION PROGRAM
INSTITUTIONAL REVIEW BOARDS

To:	DAVID WHITTINGHILL KNOY
From:	JEANNIE DICLEMENTI, Chair Social Science IRB
Date:	12/15/2014
Committee Action:	**Exemption Granted**
IRB Action Date:	12/15/2014
IRB Protocol #:	1411015457
Study Title:	Teaching Introductory Game Development with Unreal Engine: Challenges, Strategies, and Experiences

The Institutional Review Board (IRB) has reviewed the above-referenced study application and has determined that it meets the criteria for exemption under 45 CFR 46.101(b)(1) .

If you wish to make changes to this study, please refer to our guidance "**Minor Changes Not Requiring Review**" located on our website at http://**www**.irb.purdue.edu/policies.php. For changes requiring IRB review, please submit an **Amendment to Approved Study** form or **Personnel Amendment to Study** form, whichever is applicable, located on the forms page of our website **www**.irb.purdue.edu/forms.php. Please contact our office if you have any questions.

Below is a list of best practices that we request you use when conducting your research. The list contains both general items as well as those specific to the different exemption categories.

General

- To recruit from Purdue University classrooms, the instructor and all others associated with conduct of the course (e.g., teaching assistants) must not be present during announcement of the research opportunity or any recruitment activity. This may be accomplished by announcing, in advance, that class will either start later than usual or end earlier than usual so this activity may occur. It should be emphasized that attendance at the announcement and recruitment are voluntary and the student's attendance and enrollment decision will not be shared with those administering the course.
- If students earn extra credit towards their course grade through participation in a research project conducted by someone other than the course instructor(s), such as in the example above, the students participation should only be shared with the course instructor(s) at the end of the semester. Additionally, instructors who allow extra credit to be earned through participation in research must also provide an opportunity for students to earn comparable extra credit through a non-research activity requiring an amount of time and effort comparable to the research option.
- When conducting human subjects research at a non-Purdue college/university, investigators are urged to contact that institution's IRB to determine requirements for conducting research at that institution.
- When human subjects research will be conducted in schools or places of business, investigators must obtain written permission from an appropriate authority within the organization. If the written permission was not

Appendix B Survey

1. What is your gender?

#	Answer		Response	%
1	Male		15	88%
2	Female		2	12%
	Total		17	100%

Statistic	Value
Min Value	1
Max Value	2
Mean	1.12
Variance	0.11
Standard Deviation	0.33
Total Responses	17

2. What year are you in?

#	Answer		Response	%
1	Freshman		0	0%
2	Sophomore		0	0%
3	Junior		10	59%
4	Senior		7	41%
5	Graduate		0	0%
	Total		17	100%

Statistic	Value
Min Value	3
Max Value	4
Mean	3.41
Variance	0.26
Standard Deviation	0.51
Total Responses	17

3. What is your major?

Text Response
Computer Graphics Technolgy
CGT
CGT
Computer Graphics Technology
CGT
Computer Graphics Technology
CGT
Computer Graphics Technology
Computer Graphics Technology
CGT
CGT
CGT
Computer Graphics Technology
CGT
cit
CGT
computer graphics technology

Statistic	Value
Total Responses	17

4. CGT 241, or its modeling equivalent, was useful in completing this course:

#	Answer		Response	%
1	Strongly Agree		6	35%
2	Agree		4	24%
3	Somewhat Agree		3	18%
4	Neither Agree nor Disagree		3	18%
5	Somewhat Disagree		0	0%
6	Disagree		0	0%
7	Strongly Disagree		1	6%
	Total		17	100%

Statistic	Value
Min Value	1
Max Value	7
Mean	2.47
Variance	2.64
Standard Deviation	1.62
Total Responses	17

5. CGT 215, or its programming equivalent, eas useful in completing this course:

#	Answer		Response	%
1	Strongly Agree		2	12%
2	Agree		5	29%
3	Somewhat Agree		1	6%
4	Neither Agree nor Disagree		6	35%
5	Somewhat Disagree		1	6%
6	Disagree		1	6%
7	Strongly Disagree		1	6%
	Total		17	100%

Statistic	Value
Min Value	1
Max Value	7
Mean	3.35
Variance	2.87
Standard Deviation	1.69
Total Responses	17

6. What types of modeling, coding, or game software were you familiar with before this course?

Text Response
Modeling - Maya Coding - C and Java
Maya 2013, java programming language
I had not taken a 3D modeling class, however was taking CGT 241 alongside this class. I have had experience with C#, Java, Lua and minimal Unity.
I had a fair amount of modeling experience in Maya and I knew some coding in C++. I had also some online tutorial experience in Unity.
Maya, Visual Studio, Unity
Autodesk Maya (2013 & 2014) and Microsoft Visual Studio (C++)
maya
Maya
I took CGT241 in tandem, which made things difficult early on. I'm somewhat familiar with code, as I used to be a CS major before CODOing. I've messed around with various level editors for games before, creating things but nothing huge or commercial.
Standard modeling
Maya, Catia
Maya, Unreal, programming in Java.
Maya modeling, some Python, Jython, and C+.
Modeling in maya and 3ds max
none
CNIT 105, C programming
blender,maya,skecthup,unity,3ds max,visual studio

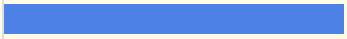
Statistic	Value
Total Responses	17

7. How difficult, on average, would you rate the exercises?

#	Answer		Response	%
1	Very Easy		0	0%
2	Easy		1	7%
3	Somewhat Easy		4	29%
4	Neutral		2	14%
5	Somewhat Difficult		3	21%
6	Difficult		2	14%
7	Very Difficult		2	14%
	Total		14	100%

Statistic	Value
Min Value	2
Max Value	7
Mean	4.50
Variance	2.58
Standard Deviation	1.61
Total Responses	14

8. What was the easiest exercise?

#	Answer		Response	%
1	Art (modeling, texture, etc.)		10	71%
2	Programming		2	14%
3	Design		2	14%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	3
Mean	1.43
Variance	0.57
Standard Deviation	0.76
Total Responses	14

9. Why?

Text Response
Know how to model better than programming and the design process was already laid out for us in exercises
Most familiar with how to do it.
Art styles can vary damatically and there's always a wide range of acceptable results.
It was the easiest for me because I already had a decent amount of modeling and texturing experience.
You can style it the way you want
I am familiar with Autodesk Maya, so I can model (inorganic) models pretty quickly as well as texture them . I am also slightly more on the artistic side versus programming. However, I am also fairly new to programming, so I have not been able to code very much (outside of the courses).
I learned it before
I like to design
Art requires skills such as modeling that I was not developed with. Programming portions were step-by-step, so they were easy if time consuming when you did things right. But if you did things wrong, due to the fact that you didn't understand the underlying logic behind quite everything you were doing and the big codebases you were building on, it could be hard to find a solution. Blueprints helped, but Unreal 4's newness hurt, so it balanced out.
Unreal didn't work for me
Because it was pretty much step by step on this, the only time it was an issue was when something was mistyped in the instructions and I had to determine what it was supposed to be.
Because Maya actually works on the lab computers.
major related
Both the in-engine tools and external programs made art fairly easy. Design wasn't bad either.

Statistic	Value
Total Responses	14

10. What was the hardest exercise?

#	Answer		Response	%
1	Art (modeling, texture, etc.)		1	7%
2	Programming		12	86%
3	Design		1	7%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	3
Mean	2.00
Variance	0.15
Standard Deviation	0.39
Total Responses	14

11. Why?

Text Response
What programming skills I have are too simple to comprehend gaming programming
Unaware of the language used.
I didn't much care for Unreal's blueprint system and I've yet to delve into C++. Compared to other programming paradigms it seemed odd.
I didn't have much in the way of programming knowledge.
I was starting from scratch, I didn't know anything beforehand
As previously mentioned, I am pretty new to programming, so I do not have much experience in coding prior to CGT 215. Additionally, some of the coding for Unreal is complicated and/or difficult to find.
I do not have much knowleage about it
I'm really bad at programming
See above.
Unreal didn't work for me
I wouldn't really call it that hard, it was just overall the hardest I suppose by just a little bit.
Because Unreal doesn't work on the lab computers.
never touched before
For me, there was a considerable jump in having taken CNIT 105 and CGT 215 for programming classes, then trying to apply that head-on in a real game engine making a project for CGT 345. There was a real disconnect between the first two classes, and this one. There does need to be a programming class in-between that helps enable code experience to be applied to a professional game engine.

Statistic	Value
Total Responses	14

12. Which exercises were the most useful?

Text Response
The exercises meant to be done in lab, because visual studio never worked in lab or at home
All were equally useful.
Being introduced to the different features in Unreal were helpful.
I really thought the exercises which taught us how to bring in 3D models with their textures and place them inside of the Unreal world were the most useful for me.
Exercise 2
The introduction to Unreal (first couple of exercises) as well as any of the modeling exercises were the most useful.
Programming
Modeling ones and implementing
The most useful exercises were the ones that dealt with functions of Unreal (or Unity) rather than creating assets for them. This is because the exercises that were just modeling, texturing or so on didn't actually teach anything new beyond how to import things into Unreal, which is mostly self-explanatory.
Unity ones
I'd like to think all of them were.
The terrain modeling ones
whole set
The big project itself was by far the most useful, with the exercises often pointing to a way to do something, sometimes better than what you had tried before. Exercises should be about exploring what's possible, a bit more than how to do it.

Statistic	Value
Total Responses	14

13. What steps, if any, would you recommend to improve the course exercises?

Text Response
More images of the blueprints
Make sure the steps are complete and there's no missing or incomplete steps.
I would suggest proving more explanation of each step in the labs (eg, why we are doing what we're doing, how we will use it, where this step is common in games, etc). I would much rather have made less progress with a game and spent more time developing a sturdy foundation to begin learning from. I feel one can't really teach a game design engine as much as they can provide a solid base for students to build off of and learn how to teach themselves.
A bit more of explanation for why and how the programming and the node placement and connections are working would be nice. It felt like we were really just following step by step and hoping that things were going to work and if things went wrong we really didn't know how to fix it.
Stick with one program that both Prof and TA know well, and get more Unity licenses for the lab
Be sure to include more screenshots of the coding portions; I had trouble with some of the more difficult coding parts, especially when there was not a screenshot on what our code should look like. Also, some more exercise demos would be helpful, especially for some of the later exercises in addition to the first couple exercises.
Tell students how each steps work
None
Explain in more detail why certain things are being done in the instructions, so it's not just a bullet point list. Not much explanation is needed, but some context particularly in complicated code sections or similar would help students to fix problems if they did something wrong. I was also disappointed that there wasn't more (any, really) on the principles of game design, level design or such. Perhaps that is intended for the follow-up course, and/or perhaps this course is merely meant to address the technical knowledge of how to do things in Unreal. I would've liked some, for what it's worth.
Actually teach course lessons, don't use an engine that doesn't compile.
Proofread things so that the person doing the exercises doesn't have to fumble around confused because something they were told to do doesn't exist or is the wrong thing.
Choose a game engine that actually works
it's good overall
Sometimes, trying to follow instruction was difficult. There were also times when if something was done incorrectly in a previous exercise, or at least done differently, later exercises became far more difficult. Exercise 9 was especially long and tough if not all of the previous exercises were done right. A bit of smoothing of each week's exercise load, and a broad-view plan that shows each week in context may be useful. Also: alternative approaces, or at least methods to make an alternative method work with a following week's plan work may help. Lots of work in that, though.

Statistic	Value
Total Responses	14

14. The team projects went well this semester:

#	Answer		Response	%
1	Strongly Agree		1	7%
2	Agree		6	43%
3	Somewhat Agree		2	14%
4	Neither Agree nor Disagree		3	21%
5	Somewhat Disagree		2	14%
6	Disagree		0	0%
7	Strongly Disagree		0	0%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	5
Mean	2.93
Variance	1.61
Standard Deviation	1.27
Total Responses	14

15. Why?

Text Response
Not very structured but still managable
We all worked well together.
I for one had quite a solid group. We had a good distribution of talent and we worked very well together.
My groups project turned out really well, I felt but there were a few groups which didn't manage to get a whole lot done. But all in all, I did feel like the group projects were a success.
Some groups weren't completely prepared or finished because of lack of knowledge of availability of help
Overall, my group went pretty smoothly on our project with the exception of one of the group members. Because he pretty much never showed up to class after the first couple of weeks, we did not expect him to contribute much to the project, and by the time he did, it was already too late. With him as an exception, we managed to get our game mostly operating.
All groups did good project
Team mates did a lot
I wish we had more time to polish our product, but it all went pretty well apart from the fact that one team member stopped coming to class.
Not enough time
It was kind of disorganized from the beginning it felt like. We started off being told that we should just have ideas and then we'd decide on whether we liked the idea or not, but then it seemed like we completely skipped that and were expected to just have our ideas ready by the next meeting.
We weren't taught how to use unity.
lacking guidance
TIME. Always need more time. Until it's done. Then more time to fix the bugs.

Statistic	Value
Total Responses	14

16. The weekly milestones were helpful:

#	Answer		Response	%
1	Strongly Agree		3	21%
2	Agree		3	21%
3	Somewhat Agree		5	36%
4	Neither Agree nor Disagree		2	14%
5	Somewhat Disagree		0	0%
6	Disagree		1	7%
7	Strongly Disagree		0	0%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	6
Mean	2.71
Variance	1.91
Standard Deviation	1.38
Total Responses	14

17. All team members contributed meaningfully to the project:

#	Answer		Response	%
1	Strongly Agree		3	21%
2	Agree		5	36%
3	Somewhat Agree		5	36%
4	Neither Agree nor Disagree		0	0%
5	Somewhat Disagree		0	0%
6	Disagree		0	0%
7	Strongly Disagree		1	7%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	7
Mean	2.50
Variance	2.27
Standard Deviation	1.51
Total Responses	14

18. The challenges we faced were primarily technical or team-related

#	Answer		Response	%
1	Technical		14	100%
2	Team-Related		0	0%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	1
Mean	1.00
Variance	0.00
Standard Deviation	0.00
Total Responses	14

19. Why?

Text Response
Each team member focused on specific things, so it was difficult to learn all aspects of creating games
We didn't have any problems with our group
In my case I had a mix of both technical and team-related. I certainly had to pull a lot more weight, but at the same time I can't expect every member to be a leader of every group they're in.
Our team worked together really well but we did come across a few issues with the models as well as some code issues we had to troubleshoot.
The program was integrated in the course at the last minute, so we lost days of learning and we didn't know much of anything about the complicated engine
Unreal Engine 4 is pretty new, so there was not a lot of help/references out their for some of the features we wanted in our game (at least for me and one other group member of mine ran into this issue). However, we also had some team-related issues too, such as a consistently absent group member and a little bit lacking on the artistic side of the game development.
s
Time windows
Besides simple lack of time, most of the issues we had centered around using and being comfortable with the engine. Ignoring the one team member who left our group, we worked together fine.
Time constraints
Just getting things done and organized.
We weren't taught unity
unreal 4.3 doesn't support video assets
If a team member is not contributing meaningfully, the team's project may have to make adjustments that take the planned project's end beyond the end of the semester. This affects the overall team's grade. Then again, being able to explain what went wrong, and the steps taken to address the issue seemed to work well to keep problems such as this from becoming a real issue.

Statistic	Value
Total Responses	14

20. I would prefer the class be taught using:

#	Answer		Response	%
1	Unreal		9	64%
2	Unity		5	36%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	2
Mean	1.36
Variance	0.25
Standard Deviation	0.50
Total Responses	14

21. Why was this engine preferred?

Text Response
Unreal is obviously a powerful engine, but it is not beginner friendly, where Unity
It is apparently a desired skill by employers
I feel like most of the students in the class are going to be beginners to game development and Unity has always been said to be a good starting place. If there were to be a higher up level of the class, I would then suggest Unreal. Unity is certainly better for beginners, but Unreal is better for making a profession.
It is very unique and powerful. My group made our game using Unity because our coder knew C# but I would have much rather used Unreal for the power it has behind it.
Because the Prof knows a lot more about it, thus he can actually help out in class instead of just directing us.
For me, Unreal has much more recognition (and most likely potential) than Unity does (even if the fourth version is somewhat new), and it looks better on one's application/skill or program set. After all, this is a game engine that most of the big name game companies use.
new engine, amazing effect
Interesting
I can't say as much about Unity, but Unreal seems reasonable enough and it seems to be more the industry-standard. It just may take some time to know the engine and therefore teach the course better.
It worked
Unity was easier but I didn't get to spend too much time with it so I don't really have enough experience to say whether it would actually be easier or not.
Because unreal doesn't work on the lab computers.
new, popular, and more fun.
Price, plans of use, blueprint system vs C# only, industry recognition, preferred company. Others may apply.

Statistic	Value
Total Responses	14

22. I would have preferred a large individual semester project instead of a team project:

#	Answer		Response	%
1	Strongly Agree		0	0%
2	Agree		2	14%
3	Somewhat Agree		1	7%
4	Neither Agree nor Disagree		4	29%
5	Somewhat Disagree		2	14%
6	Disagree		3	21%
7	Strongly Disagree		2	14%
	Total		14	100%

Statistic	Value
Min Value	2
Max Value	7
Mean	4.64
Variance	2.71
Standard Deviation	1.65
Total Responses	14

23. The exercises helped to complete the team projects:

#	Answer		Response	%
1	Strongly Agree		2	14%
2	Agree		1	7%
3	Somewhat Agree		4	29%
4	Neither Agree nor Disagree		1	7%
5	Somewhat Disagree		1	7%
6	Disagree		3	21%
7	Strongly Disagree		2	14%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	7
Mean	4.07
Variance	4.38
Standard Deviation	2.09
Total Responses	14

24. How did it help or hinder?

Text Response
If the team project was a first person game, it would help. But my project was a 2D platformer, so the exercises did not help the team project.
The exercises took time away from the project.
The exercises did not apply to my team project because we used Unity. I would have liked, however, to have had the option to complete Unity exercises instead of Unreal.
The exercises dealt with many things that my group ended up using in our group project. It helped us learn how to use the program along the way, although some of us needed to know more complicated programming earlier on.
The exercises helped me on navigating through Unreal's various featuresnmainly, but it also help me with setting up a (simple) game and coding within it as well as how to use import various items within it and adjusting them as necessary.
each exercise is very important part of making game
Help to know unreal better
I wasn't our groups resident programmer, who the exercises may have helped somewhat. That being the case, the exercises had almost no bearing on my group work at all. Instead, they were just separate work I fell behind on while doing group work.
It was in unreal, which didn't work on the lab computers.
It really did neither, since I was mainly doing the artwork portion of things and therefore I ended up only needing a bit of the knowledge in order to place things in the game.
They wasted my time and frustrated me
somewhat
They took time. Often, an exercise would point out something useful after I'd already spent days takling a problem. Othertimes the exercises would simply point out an alternative. They were helpful, but sometimes overwhelming. Might need to be shifted a bit into a dedicated programming course to be taken before this class.

Statistic	Value
Total Responses	14

25. What would you recommend, if anything, to improve the technical side of the semester project?

Text Response
Better explanations to the code and what it does.
I don't know
Just make sure that the software works on the computers ahead of time. I don't expect that really being applicable to future semesters though.
Just get Unreal to work in the labs, please.
It would have been better if the Prof looked into the engine more to see if it would have worked with the lab hopefully finding the problems earlier on
Give more in class demos, particularly on the more complex/difficult portions, and include screenshots for them (at least for complex coding).
no
None
Maybe try to explain how to use the engine more broadly? Any type of structure to the group projects would also probably help. Learning through experience is nice and all, but throwing students at a relatively unknown engine and telling us to make literally any type of game was broad. We got feedback on our milestones but we had to make those as well, and as mentioned the exercises didn't always help. Creative freedom is great, but some limitations could fuel creativity in addition to making things easier to manage.
Use unity for exercises
I suppose it's good as it is currently.
Choose a game engine that works on the lab computers.
giving more guidance
More time exploring how to make things work, and why things need to be done in certain ways, and how to select when to use a blueprint system vs. creating fresh code, etc.
Points again to a new programming class.

Statistic	Value
Total Responses	14

26. What would you recommend, if anything, to improve the teamwork aspect of the semester project?

Text Response
Have the team project be a specific type of game (3D first person, 2D platformer) so that there would not be so many different bugs to work out in each project.
None
It might be nice to have students divide themselves into a subgroup and divide into groups from there. This way you would have an even distribution of programmers, writers, and artists on each team.
Nothing that I can think of. My group worked pretty well.
Allowing people to work in smaller groups if they wanted so that they didn't have to work in groups and make games that they didn't enjoy
Make sure it is clear that when the class is to get into groups for the first time that they know it is highly probable that those individuals will be one's group members (this happened to my group but everything mostly worked out fine).
no
None
I'm not sure how you could improve teamwork between random classmates. More guidelines as mentioned before could help, as teams could more accurately know and divvy up what they had to do.
Nothing
Better help keep things organized from the start, give people the idea of the list of what each person should do every week from the beginning so everyone is on track.
N/A
random assign
Having flexibilty to make massive changes as issues develop was an important aspect in making the project more or less "successful". I can't think of much beyond that.

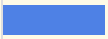
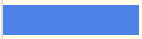
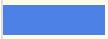
Statistic	Value
Total Responses	14

27. What topics do you feel should be covered in lecture?

Text Response
Coding concepts
We didn't really have lectures
Success of companies How to find work Current events Quality of life for game designers Different components of game design within the industry
I would have loved to learn more about the different things we were doing inside of Unreal.
The main aspects of important programming in video games, such as level design, GUIs, and possible steps of interaction
A brief coding and/or modeling tutorial for anyone that somehow skipped CGT 215 or 241/340. Otherwise, just lecture on how to use various features of Unreal (or whatever the game engine is) as well as how to think of a game idea/concept and/or the various (but basic or "standard") gaming genres that could be used or made for the class.
programming and some design
Not sure
1. Clarifying/explaining/contextualizing lab work, 2. General game design principles, 3. Maybe a little bit on how jobs in the industry work?
Unity
I'm not sure what else should be covered.
Unity development
we don't have lecture
explanations of technical aspects of using a game engine, and what's possible and how to use it.

Statistic	Value
Total Responses	14

28. I prefer the lab-oriented format of the class to a traditional lecture course:

#	Answer		Response	%
1	Strongly Agree		3	21%
2	Agree		4	29%
3	Somewhat Agree		4	29%
4	Neither Agree nor Disagree		0	0%
5	Somewhat Disagree		3	21%
6	Disagree		0	0%
7	Strongly Disagree		0	0%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	5
Mean	2.71
Variance	2.07
Standard Deviation	1.44
Total Responses	14

29. I think the ideal distribution of lab work versus lecture for this course should be:

#	Answer	Min Value	Max Value	Average Value	Standard Deviation
1	Lecture	0.00	70.00	19.79	21.02
2	Lab	2.00	90.00	40.71	34.32

30. I am confident with my game development skills:

#	Answer		Response	%
1	Strongly Agree		0	0%
2	Agree		2	14%
3	Somewhat Agree		8	57%
4	Neither Agree nor Disagree		2	14%
5	Somewhat Disagree		0	0%
6	Disagree		0	0%
7	Strongly Disagree		2	14%
	Total		14	100%

Statistic	Value
Min Value	2
Max Value	7
Mean	3.57
Variance	2.42
Standard Deviation	1.55
Total Responses	14

31. I am confident with my Unreal Engine skills:

#	Answer		Response	%
1	Strongly Agree		0	0%
2	Agree		0	0%
3	Somewhat Agree		6	43%
4	Neither Agree nor Disagree		3	21%
5	Somewhat Disagree		2	14%
6	Disagree		1	7%
7	Strongly Disagree		2	14%
	Total		14	100%

Statistic	Value
Min Value	3
Max Value	7
Mean	4.29
Variance	2.22
Standard Deviation	1.49
Total Responses	14

32. I am now motivated to pursue game development:

#	Answer		Response	%
1	Strongly Agree		3	21%
2	Agree		3	21%
3	Somewhat Agree		4	29%
4	Neither Agree nor Disagree		1	7%
5	Somewhat Disagree		1	7%
6	Disagree		2	14%
7	Strongly Disagree		0	0%
	Total		14	100%

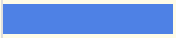
Statistic	Value
Min Value	1
Max Value	6
Mean	3.00
Variance	2.92
Standard Deviation	1.71
Total Responses	14

33. Are you interested in taking the advanced course?

#	Answer		Response	%
1	Yes		9	64%
2	No		5	36%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	2
Mean	1.36
Variance	0.25
Standard Deviation	0.50
Total Responses	14

34. You had a good overall experience with the class:

#	Answer		Response	%
1	Strongly Agree		0	0%
2	Agree		7	50%
3	Somewhat Agree		5	36%
4	Neither Agree nor Disagree		0	0%
5	Somewhat Disagree		0	0%
6	Disagree		0	0%
7	Strongly Disagree		2	14%
	Total		14	100%

Statistic	Value
Min Value	2
Max Value	7
Mean	3.07
Variance	2.99
Standard Deviation	1.73
Total Responses	14

35. Why?

Text Response
Fun class, results are clear and straight forward
It was fun and exactly what I want to do with my life.
It was neat to be able to see what the other students brought to the class and speak with other people that had the same interest as me.
I just really enjoyed how this class worked and what we really had done in the group projects.
It was interesting learning a more modern engine that was a bit more complicated than the rest.
I was able to learn how to make a game and face typical developing issues (particularly with coding),, as well as work with a group to make a game, even if it was simple. I can officially say I contributed to make a game.
we can do project at home
I like game designing
I've wanted to work on games, and this class let me, so that was expectedly enjoyable. It could've been managed a bit better though, and my lack of experience in other areas like modeling made things a bit rough here and there.
There was hardly any teaching, and the engine used in the labs was broken
It was good but sometimes it was hard to complete the exercises because something was mistyped or what I was told to do wasn't correct so I had to get the problem fixed in order to finish the exercise and do the next one.
They chose an engine that didn't function properly on the lab computers, causing me to be unable to finish the exercises.
enjoyed
I actually found myself having fun while working on the project for this class. I absolutely abhorred the web development class.

Statistic	Value
Total Responses	14

36. The technical issues in the class did not affect the educational experience:

#	Answer		Response	%
1	Strongly Agree		0	0%
2	Agree		2	14%
3	Somewhat Agree		1	7%
4	Neither Agree nor Disagree		2	14%
5	Somewhat Disagree		7	50%
6	Disagree		0	0%
7	Strongly Disagree		2	14%
	Total		14	100%

Statistic	Value
Min Value	2
Max Value	7
Mean	4.57
Variance	2.26
Standard Deviation	1.50
Total Responses	14

37. How well was the class taught?

#	Answer		Response	%
1	Very Good		1	7%
2	Good		8	57%
3	Fair		3	21%
4	Poor		1	7%
5	Very Poor		1	7%
	Total		14	100%

Statistic	Value
Min Value	1
Max Value	5
Mean	2.50
Variance	1.04
Standard Deviation	1.02
Total Responses	14

38. What advice do you have to improve the course?

Text Response
More coding lessons
Fix the technical issues in the labs. Make the exercise more streamlined
Put more emphasis on creating a solid base for learning and less on creating a finished product. As corny as this sounds, this class should be somewhat like a nest, from which the students can learn to fly on their own after it's over.
Make sure that the program works in the lab and make sure that the exercises don't have as many issues with them.
Get more Unity Licenses and test out an engine in the lab before adding it to a course
More in class demos, especially for the introduction exercises and difficult coding portions of an assignment. Stick to only one gaming engine, unless the other one(s) are used for more than one or two assignment.
teaching programming
None
As said before, more guidelines and explanation would help. I would enjoy more talk on game design and how to make games beyond the bare bones technical aspect (design is somewhat opinion, but it has some generally accepted points; and I'm sure professional games have a lot in common with how they are built and organized).
Use unity
Just again make sure the instructions are written out properly.
Use Unity, and actually teach game development.
none
Known issues have already been tackled, or are in progress. Hopefully the people at Unreal will start taking more seriously the deployment of their program on University computers, if they haven't already.

Statistic	Value
Total Responses	14

39. What would have helped you personally?

Text Response
More coding lessons
Better excercises
I would have really liked just some information on what and why I am doing the things I am doing inside of Unreal.
If the Prof knew more on the engine
Screenshots for complex/difficult coding portions, especially for the last few assignments. Being able to work on an assignment in lab rather than at home while still using the same instructions in addition to being able to build a program through Unreal in lab too.
.
None
What I said above would have helped. Getting behind on exercises was my own fault, but it would've been harder if the explanations were clearer and I wasn't winging it on so much of the group project.
Not using a broken engine
Not much really, didn't have many other problems.
If the professor chose a program that actually worked.
none
Getting a better understanding of how to master using the game engine. I still feel less than competent in being an asset in a game development company.

Statistic	Value
Total Responses	13

40. Are there any other comments that you would like to make about the course?

Text Response
More coding lessons
N/A
While there were sometimes occasions in which the instructions were unclear in the assignments, I felt you did an excellent job of helping the other students when they had issues.
The TA did a great job writing the course and exercises. Good job, Nick
No further comments other than what I had previously mentioned.
.
None
Everything I could say has been covered. Hopefully the next gaming course has some of these things I wanted more of.
I regret taking this course
Not really.
This course was poorly run and an utter disappointment to myself and others.
nope
Fun, but tough at times. Really could use that programming course for those who want to take it.

Statistic	Value
Total Responses	13

Appendix C Exercises

CGT 345

Due at the end of lab

Goal

In the exercise, you will learn the basics of packaging an Unreal project.

Instructions

1. Open Unreal Engine.

2. Go to *Library*.

3. In *Engine Slots*, launch the latest version of Unreal 4.

4. Go to the *New Project* tab and scroll down and select *Blueprint Side Scroller*.

5. Above the *Create Project* button, click on the arrow and pick a destination for the project along with the name *ex00*.

6. Click on *Create Project*.

7. Got to the top of the main editor and select *Blueprints* > *Open Level Blueprint*.

8. *Right click* and type *Escape* in *Search*. Select it to place it in *Graph*.

9. *Right click* again and add *Execute Console Command*. In Command, type *Exit*.

10. Connect *Pressed* from *Escape* into *Execute Console Command*.

11. In the current versions of the Unreal Editor, the Engine Scalability settings do not transfer to the Game Settings when packaging. To change this, add the following to your *Level Blueprint*: https://answers.unrealengine.com/questions/31421/scalability-reference-does-not-apply-to-standalone.html
    i. Do not add the last node, *Print String*, to the chain.

12. *Compile* before closing the *Blueprint Editor*.

13. Make sure to *save* the project before packaging.

14. Go to *File* > *Package Project* > *Build Configuration* and select *Shipping*.

15. Go to *File* > *Package Project* > *Windows* to package the project into an exe.

Submission

Packaged projects will be checked in class.

Building a standalone exe

Make sure your first level is set in *Edit* > *Project Settings* > *Maps and Modes* > *Game Default Map*. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in *Quick Settings* > *Engine Scalability* will only affect the editor and not the package. Make sure that the *Build Configuration* is set to *Shipping* as well. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra:

1. This goes over some of the ways to edit your games engine scalability:
https://answers.unrealengine.com/questions/23023/trouble-configuring-game-settings.html
Also:
https://answers.unrealengine.com/questions/31421/scalability-reference-does-not-apply-to-standalone.html

CGT 345

Due Monday (9/8/2014)

Goal

In the exercise, you will be creating a first person controller for your island project.

Setup

1. Open Unreal Engine.

2. Go to *Library*.

3. In *Engine Slots*, launch the latest version of Unreal 4.

4. Go to the *New Project* tab and select *Blank*.

5. Above the Create Project button, click on the arrow and pick a destination for the project along with the name *Island*.
    i. Make sure that *Include starter content* is *unchecked*.

6. Click on *Create Project*.

7. Go to *File* > *Save* and save your map using as *Level*.
    i. It would be best to create a map folder inside of content to store any of the maps/levels.

8. Go to *Edit* > *Project Settings*. Once inside, go to *Maps & Modes* under *Game* and change the Editor Startup Map to your map by clicking on the arrow next to the description.

9. Click on *Set as Default* near the top right to save the settings for the project.

Create Game Mode

1. For this exercise, you will need to use a code editor.
    i. If you are working in lab, Visual Studio should open.
    ii. If you are working on a PC or Mac, you will need Visual Studio or Xcode:
        a. Xcode download:
https://developer.apple.com/xcode/downloads/

b. Visual Studio Express:
http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx
c. Visual Studio Professional Student:
https://www.dreamspark.com/Product/Product.aspx?productid=72&cmpid=W_VS_DSV_DS_728x90_ENG

2. Follow the instructions from *Creating a GameMode*:
https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial#Creating_a_GameMode

3. Any reference to *FPSProject > Source > FPSProject* is *Island > Source > Island* for you.
i. The *override* command is different depending on the version of the Unreal Engine.
a. Anything below *4.3* will use *OVERRIDE*.
b. Anything at or above 4.3 will use *override*.

Making a Character

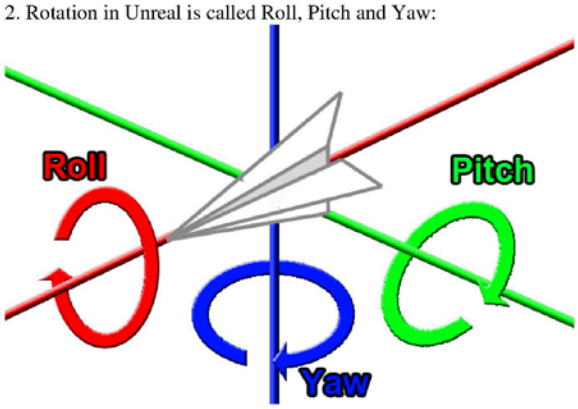
1. Follow the instructions of *Making a Character*:
https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial#Making_a_Character

2. Any reference to *FPSProject > Source > FPSProject* is *Island > Source > Island* for you.
i. The *override* command is different depending on the version of the Unreal Engine.
a. Anything below *4.3* will use *OVERRIDE*.
b. Anything at or above *4.3* will use *override*.

Movement

1. Follow the instructions from *WASD Movement* to *Jumping*, stop before *Adding a Mesh to Your Character*:
https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial#WASD_Movement

2. Rotation in Unreal is called Roll, Pitch and Yaw:



3. If you would like your character to jump higher then there is a couple things you can do.
    i. You can add this code to the constructor of *FPSCharacter.cpp*:
    ```
    CharacterMovement->JumpZVelocity = 500.0f;
    ```
    ii. You can change the Default Gravity Z by going to *Edit* > *Project Settings* and then to *Physics* underneath *Engine*.

4. Go to the top of the editor and click *Blueprints* > *Open Level Blueprint*.

5. Apply the same game settings as the previous exercise:
https://answers.unrealengine.com/questions/31421/scalability-reference-does-not-apply-to-standalone.html

6. *Right click* and type *Escape* in *Search*. Select it to place it in *Graph*.

7. *Right click* again and add *Execute Console Command*. In Command, type *Exit*.

8. Connect *Pressed* from *Escape* into *Execute Console Command*.

9. *Compile* and *Save* before closing the *Blueprint Editor*.

10. Make sure to *save* the project before *packaging*.

Result

The character should be able to move around the platform using the WASD keys and look around with the mouse. They should also be able to jump using the spacebar.

Submission

Packaged projects will be checked in class on Monday (9/8/2014).

Building a standalone exe

Make sure your first level is set in *Edit* > *Project Settings* > *Maps and Modes* > *Game Default Map*. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in *Quick Settings* > *Engine Scalability* will only affect the editor and not the package. Make sure that the *Build Configuration* is set to *Shipping* as well. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra:

1. This series will help with C++ programming, plus parts of this will be used in future exercises:
http://www.youtube.com/watch?v=vtcWrcscXos&list=PLZlv_N0_O1gaCL2XjKluO7N2Pmmw9pvhE&index=2

2. Series goes over Unreal Editor basics:
http://www.youtube.com/watch?v=QMsFxzYzFJ8&list=PLZlv_N0_O1gaCL2XjKluO7N2Pmmw9pvhE&index=67

CGT 345

Due Monday (9/15/14)

Goal

In the exercise, you will be creating an island using the landscape tool.

Setup

1. Create a copy of you previous project.
    i. Either Clone
        a. Open Unreal Engine.
        b. Go to Library.
        c. Under My Projects, click the arrow next to Open beneath your project file from the previous week.
        d. Click on Clone
        e. In Name, make sure it keeps the original project name. If it has a different name, you will be unable to recompile in the editor due to different .dll and .sln files from the project name.
        f. Click Create.
        g. Once the clone appears, click the Open tab beneath the project.
    ii. Or copy the folder to have a backup version.
    iii. Make sure the project name is the same. If it is different from the .sln or .dll files then you will be unable to recompile code in the editor.

Materials

1. In the Scene Outliner, go to the SM_Template_Map_Floor actor. To the left of the name is an eye. Click this eye to hide the object in the scene.

2. Go to the Content Browser and right click on the Game folder.

3. Click on New Folder and name it Materials.

4. Right click again on the Game folder and create a New Folder called Textures.

5. Open the Textures folder and right click and select Import to.

6. Import the textures provided to you for the assignment.

6. Open the Material folder and right click to select Material.

7. Name this material M_Terrain.

8. Double click M_Terrain to open up the material editor.

9. Right Click in the editor and type LandscapeLayerWeight into search.

10. Select LandscapeLayerWeight and create four copies by selecting the layer and pressing Ctrl + W or copying and pasting.

11. Select each layer and name them Grass, Beach, Cliff, and GrassRock.

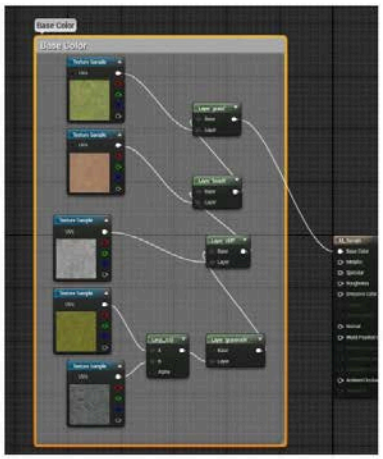
12. Drag and drop T_Ground_Grass_D, T_ground_Moss_D, T_Rock_Basalt_D, T_Rock_Sandstone_D, and T_Rock_Slate_D.

13. Drag the top right node, output, of each texture into the Layer slot.
    i. Drag T_Ground_Grass_D into *Layer 'Grass'*.
    ii. Drag T_Rock_Sandstone_D into Layer *'Beach'*.
    iii. Drag T_Rock_Slate_D into *Layer 'Cliff'*.

14. For *Layer 'GrassRock'* mix T_ground Moss_D and T_Rock_Basalt_D.
    i. Add Lerp to the editor by pressing L and left clicking.
    ii. Drag the output of the two textures into A and B of Lerp.
    iii. Drag the output of Lerp into the Layer of *Layer 'GrassRock'*.

15. Drag the output of each landscape layer into the Base and finally into Base Color of M_Terrain.
    i. Drag the output of *Layer 'GrassRock'* into the Base of *Layer 'Cliff'*.
    ii. Drag the output of Layer *'Cliff'* into the Base of Layer *'Beach'*.
    iii. Drag the output of Layer *'Beach'* into the Base of Layer *'Grass'*.
    iv. Drag the output of Layer *'Grass'* into the Base Color of M_Terrain.

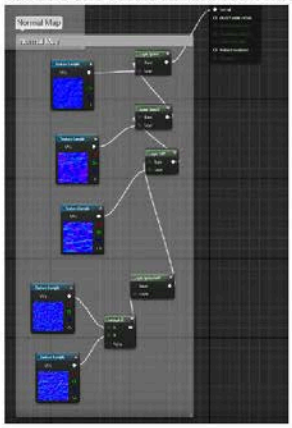
16. The End result should look similar to the below image:



17. Copy the Layers and bring in the normal map textures T_Ground_Grass_N, T_Ground_Moss_N, T_Rock_Basalt_N, T_Rock_Sandstone_N, and T_Rock_Slate_N.

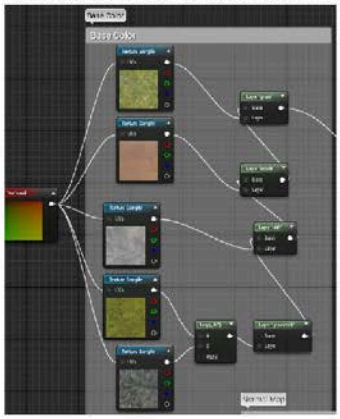
18. Repeat steps 13 to 15 except instead of plugging into Base Color, plug into Normal of M_Terrain.

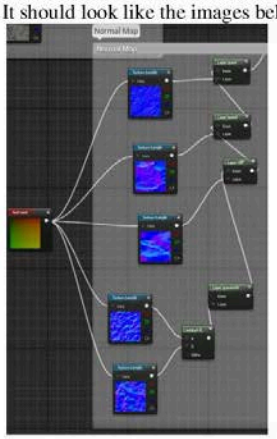
19. The end result should look like the below image:



19. Right click in an open space and type Texture Coordinates into search. Select Texture Coordinates.

20. Texture Coordinates controls the UV's for different textures. Change UTiling and VTiling to .2 and connect it to the Textures and Normal Maps. It should look like the images below.





20. Save the Material and exit the Material Editor.

Sculpting

1. In the Modes panel, select the Mountain shaped, Landscape, button.

2. Under New Landscape drag the M_Terrain into Material.

3. Scroll down and select create

4. Select Paint

5. Scroll down to Target Layers and select Beach.

6. Select the plus sign at the left most part of Beach, called Create Layer Info. Select Weight-Blended Layer (normal) from the drop down menu.

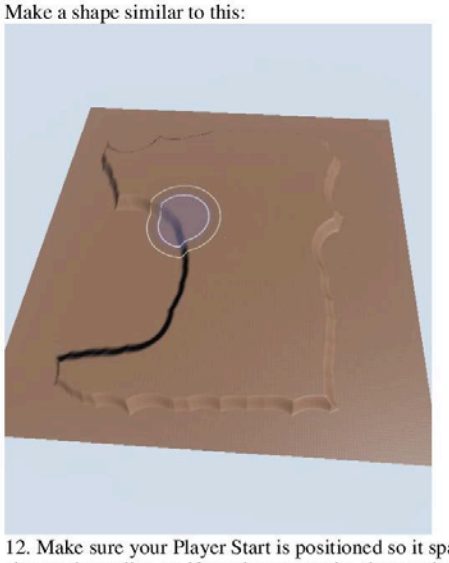
7. Repeat steps 5 and 6 for each layer.

8. Select Beach and change Brush Falloff to 0 and increase Brush Size, under Brush Settings. Ctrl + Left Click and paint the entire terrain the Beach texture.

9. Select Sculpt.

10. Go to Brush Settings and change the Brush Falloff to 0.3.

11. Hold down Shift + Ctrl+ Left Click to lower the terrain, Ctrl + Left Click raises the terrain. Make a shape similar to this:



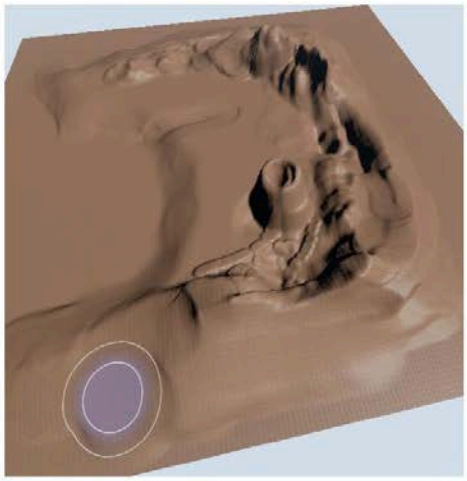
12. Make sure your Player Start is positioned so it spawns on top of the terrain. This could change depending on if you lower or raise the terrain.

13. If the edges of the island are too steep for the character, the Smooth Tool can be used to smooth out the terrain. Under Landscape Editor, Select Sculpt Tool and go down to Smooth. You can also use the Ramp Tool to make a proper incline.

14. Experiment with the different brushes and create your own types of hills and depressions. Make sure to leave some flat areas for later.

15. Create a Volcano somewhere on the island. Go back to the Sculpt Tool and change the Brush Size to 3300 with a Tool Strength of 1. Also, you may want to use a Brush Falloff of 0.5.Click on an open area until you get a height you are satisfied with.

14. Select a smaller Brush Size, 1200, a larger Brush Falloff, 0.7, and Shift + Ctrl + Left Click at the center of the hill to lower the center. This will create a volcano on your map that should look similar to this:



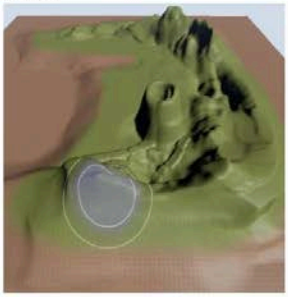
Texturing

1. Select Paint at the top of Landscape.

2. Change the Brush Falloff and Brush Size to values that you are comfortable painting with. Tool Strength will decide the opacity of the paint.

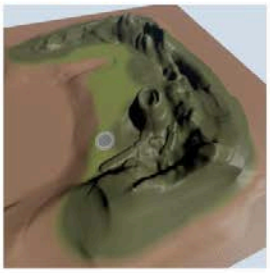
3. Select Grass under Target Layers and paint over parts of the island leaving some beach areas near the water.

i. EX:



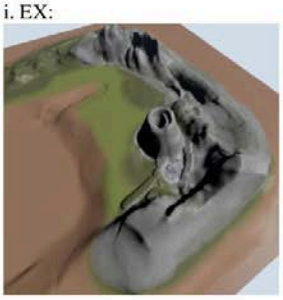
3. Use the GrassRock to paint any low hills created.

i. EX:



4. Use Cliff to paint any mountainous terrain.

i. EX:



Foliage

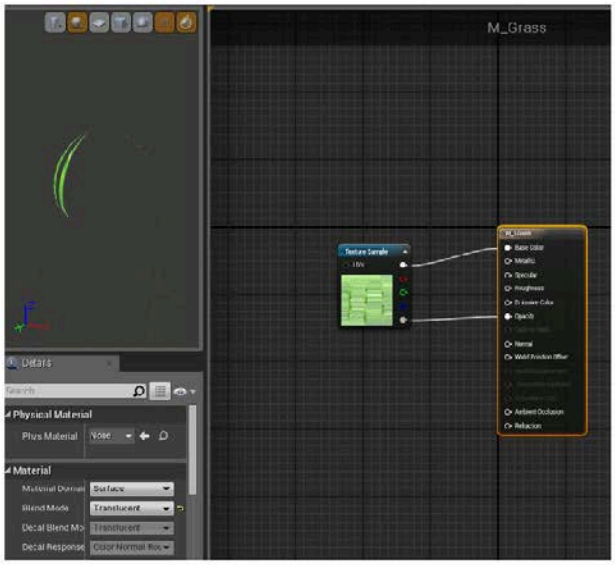
1. Under Game, create a new folder called Grass.

2. Import grass.fbx into the Grass folder. Make sure Import Material is Checked under Advanced. Select Yes when it asks if you would still like to create the texture even though it isn't a power of two.

3. Open M_Grass and select the main node in the Material Editor.

4. In the left hand corner under Material, change the Blend Mode to Translucent.

5. Drag the lowest slot of the Texture Sample into Opacity of the main node. The end result should look like this:

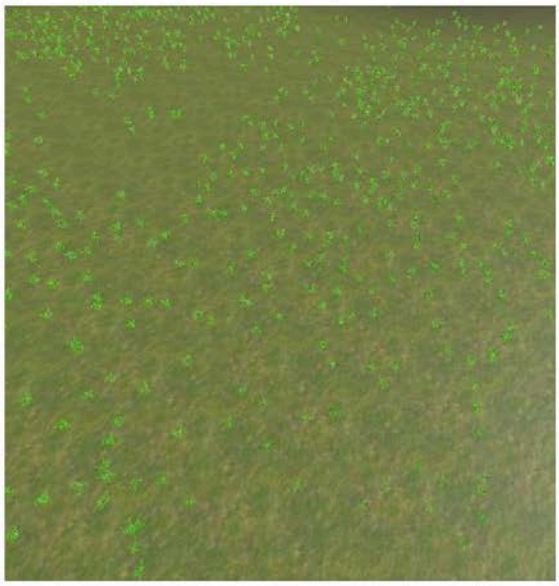


6. Save and Close the Material Editor.

7. Go to the Foliage in the Modes Tab.

8. Drag the Grass mesh into the specified region.

9. Change the Paint Density to .7 and paint the Foliage over the grass textured areas.
    i. Ctrl+Left Click to Paint and Ctrl+Shift+Left Click to erase.
    ii. Official Documentation:
    https://docs.unrealengine.com/latest/INT/Engine/Foliage/index.html
10. The end result should look something like this:



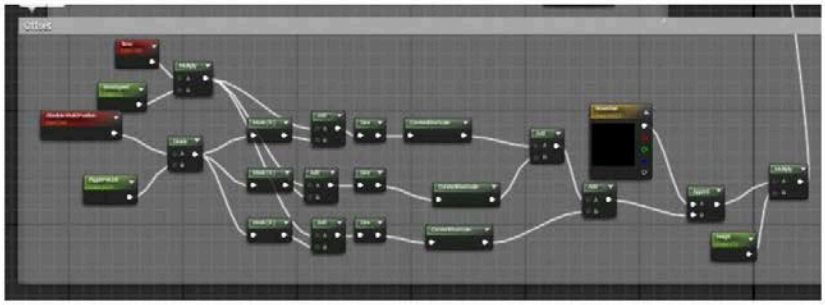
Water

1. Create a new Material called M_Ocean in the Material folder. Open M_Ocean.

2. Follow the directions in this video: https://www.youtube.com/watch?v=KPmRV1Z9ikY
    i. Use T_Rock_Slate_N.
    ii. The node for clamp is Component Mask.
    ii. If a plane is unavailable, Use SM_Template_Map_Floor as the ocean.
        a. Change the Scale, under Transform, to 100, 100, 50.
        b. In Materials, change the material to M_Ocean.
        c. Go to Collision and change the Collision Presets to NoCollision.
        d. If it is still hidden make sure the eye is open in the Scene Outliner.
    iii. Change the Fade Distance of the Depth Fade connected to Opacity to 300 instead of 100.
        a. You can also change the value of the Fresnel to have the water appear less transparent.
    iv. Multiply the Base Color value by 3 instead of .1.

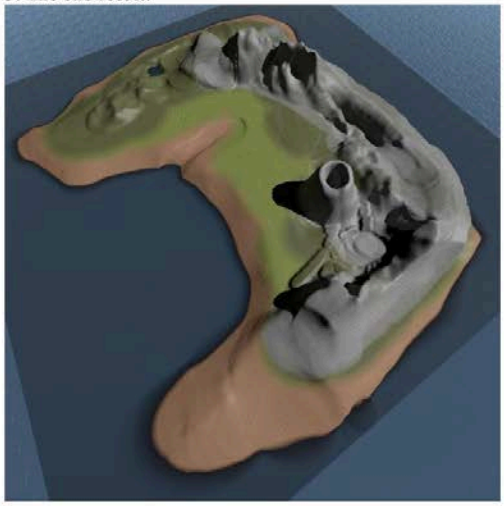
3. Follow the directions in this video: https://www.youtube.com/watch?v=GRnKO1z4WZs
    i. When it asks to make a Component Mask for R and G, make one for B as well.
    ii. Do the same process as R and G for B but with a Sine Period of 1.
    iii. ConstantBiasScale should be set to .25 for Bias and .5 for Scale.

iv. Add the Mask B line to the Add of R and G.
v. Create a 3 vector instead of a 2 vector.
vi. It should look similar to this:



4. Save and exit the Material Editor.

5. The end result:



Result

There should be an island with different types of textures and terrain as well as a volcano. This island is surrounded by water while also housing foliage.

Submission

Packaged projects will be checked in class on Monday (9/15/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Make sure that the Build Configuration is set to Shipping as well. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra:

1. For further help on the Landscape tool, you can read the official documentation, https://docs.unrealengine.com/latest/INT/Engine/Landscape/index.html, or watch this tutorial, http://www.youtube.com/watch?v=FHVuVVHlUmM.

2. For further description on creating the grass look here:
http://www.youtube.com/watch?v=Pqb76OXSQSY
3. If you would like to create moving grass, look here
http://www.youtube.com/watch?v=HhUK_02vSyE

4. Here is the official documentation for the foliage tool:
https://docs.unrealengine.com/latest/INT/Engine/Foliage/index.html

5. How to make swimmable water: http://www.youtube.com/watch?v=LtyXjSb1P-4

CGT 345

Due Monday (9/22/14)

Goal

Model and UV unwrap a house and other objects to be placed on the island

Setup

1. Create a copy of you previous project.
    i. Either Clone
        a. Open Unreal Engine.
        b. Go to Library.
        c. Under My Projects, click the arrow next to Open beneath your project file from the previous week.
        d. Click on Clone
        e. In Name, make sure it keeps the original project name. If it has a different name, you will be unable to recompile in the editor due to different .dll and .sln files from the project name.
        f. Click Create.
        g. Once the clone appears, click the Open tab beneath the project.
    ii. Or copy the folder to have a backup version.
    iii. Make sure the project name is the same. If it is different from the .sln or .dll files then you will be unable to recompile code in the editor.

Model the Outpost

1. We need a building for our player to explore on the island. It should be a relatively small building that has room for a table.

2. Using Maya (or another 3D modeling application you are familiar with), model the outpost. It should have the following features:
    i. A door
    ii. A light on the outside above the door
        a. This could be modeled into the building, or be a separate model
    iii. A small power generator next to the door
        a. Leave a flat area (display) on it so that we can apply the proper texture later on
    iv. A window
        a. Make sure this has geometry for glass, and not just an empty hole.

v. Make sure all objects are scaled correctly. The default cube would look normal sized in the editor if it is at a scale of above 300 for each axis.

3. VERY IMPORTANT: Make sure you model the inside and outside of the building, and make sure that the normals are facing the right direction. Otherwise your object will appear invisible in Unity.

i. To preview what your building will look like in Unreal, select the object and go to Display > Polygons > Backface Culling. Select the option again if you want to turn it off.
ii. To see the direction normal are facing, select the object and go Display > Polygons > Face Normals. Select the option again if you want to turn it off.

Model the rest of the objects

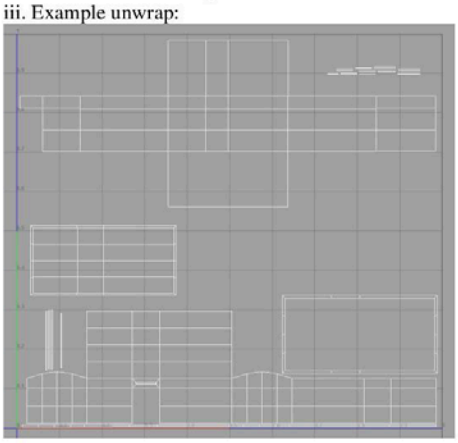
1. Inside the outpost, create 3 models:

i. A table
a. Put this near the window, so the player can see it from the outside.
ii. A ceiling light. Make sure to include separate geometry for the light bulb
iii. A box of matches
iv. This should be relatively small and placed on the table, clearly visible.

UV Unwrap the objects

1. Using Planar Mapping (or other projection mapping / UV programs, like Roadkill), UV unwrap each object.

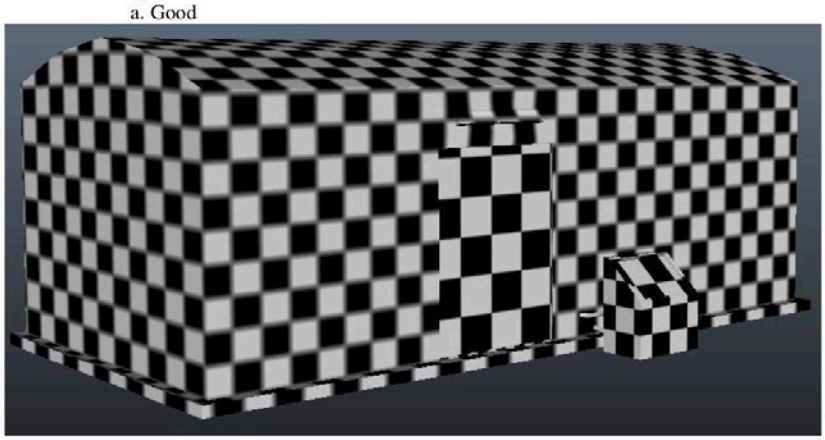
i. After unwrapping the objects, go into the UV Texture Editor and make sure all of the UV maps in the 0-1 space.
ii. Make sure that there is no overlapping UV's. If there is any overlapping, the lightmap created in the Unreal Engine will be distorted.
iii. Example unwrap:



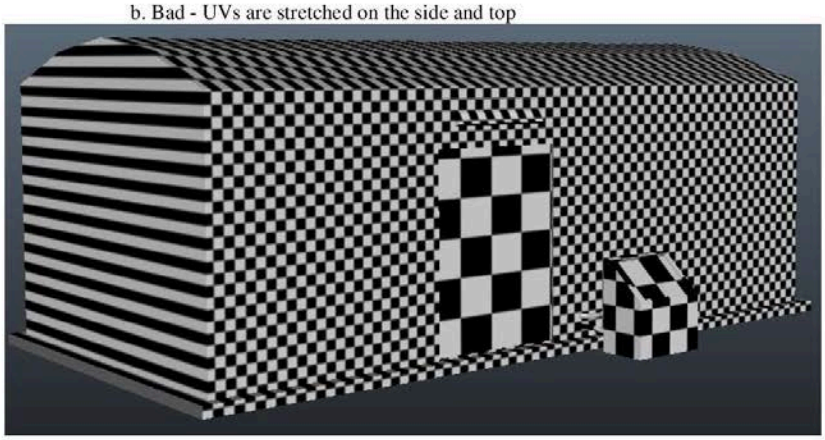
2. Create the checker material
    i. Bring the checker_UV and checker_UV_Small (from the assets provided) into your Maya project's sourceimages folder.
    ii. Open up the Hypershade (Window>Rendering Editors>Hypershade).
    iii. Create a new Lambert Material (Create>Materials>Lambert).
    iv. Double click the new Lambert material to open up its properties in the Attribute Editor.
    v. Rename the material to CheckerUV.
    vi. Click the black and white checkered box next to the "Color" attribute.
    vii. Select "File" in the dialogue box that appears.
    viii. Now, in the Image Name field that appears, click the folder icon and browse for the checker_UV.jpg file in the sourceimages folder.
    ix. Repeat steps iii through viii to make the checkerUVsmall material (using the small texture uv file).

3. Apply the checker material
    i. Bring up the Hypershade window, but keep the model visible in the perspective view as well.
    ii. Middle-mouse drag the checkerUV material onto each model. If the checker pattern appears too large, try using the checkerUVsmall material instead.

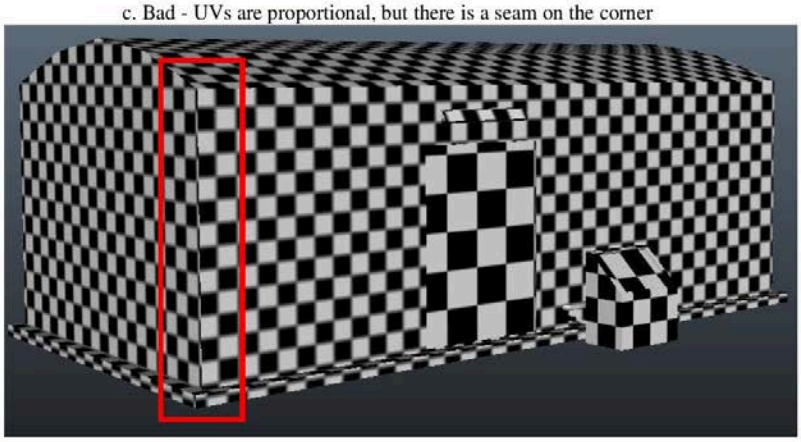
4. Check for any UV problems
    i. Now that we have a checker pattern on the objects, it will become very obvious if we have UV problems. The checker pattern should appear even and non-stretched on every object. The size of the checkers do not matter, so long as the pattern is consistent across each individual object.
    ii. Examples:
        a. Good



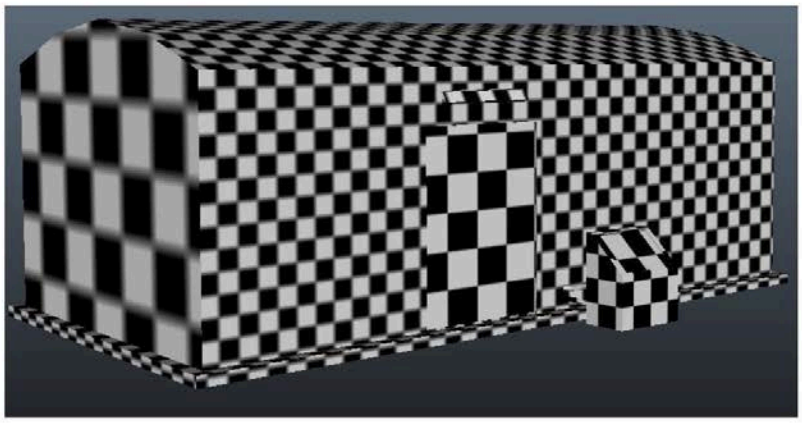
b. Bad - UVs are stretched on the side and top



c. Bad - UVs are proportional, but there is a seam on the corner



d. Bad - UVs are not proportional across the model



5. After thoroughly checking the models, fix any UV problems.

Bring the Model into Unreal

1. Select all models in the scene and delete the history with Edit > Delete By Type > History.

2. Open the Outliner (Window > Outliner) to view a list of each object. Select each object and name it.

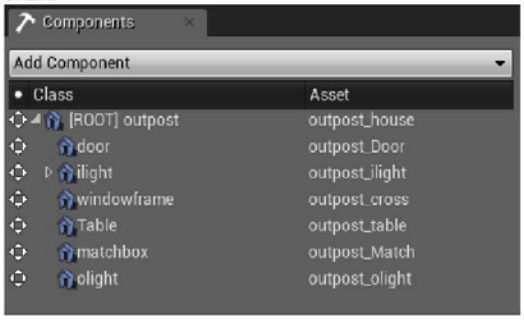
3. Organize Outliner
    i. Select all objects and group them with Ctrl + G.
    ii. Rename the group to outpost.

4. Export Models
    i. Load the FBX plugin by going Window > Settings/Preferences > Plug-In Manager. Find fbxmaya and check Loaded.
    ii. Select outpost in the Outliner.
    iii. Go to File > Export Selection.
    iv. Choose FBX Export for the Files of type field.
    v. Name the file outpost.fbx.
    vi. Select Export Selection.
    vii. Follow the same process to export a door.fbx and matches.fbx from the door and box of matches meshes.

5. Import files into Unreal
    i. Create a folder in the Game folder called Models. Import outpost.fbx into this folder.
        a. Make sure the Import Materials is checked in Advanced when importing the meshes.

6. Go to the Blueprints folder and create a blueprint called BP_House.
    i. Use the Standard Classes Actor for the Pick Parent Class option.

7. Double click BP_House to open the editor.
    i. In the Components tab use Add component to add a Static Mesh and name it outpost.
        a. In the Static Mesh Section outpost, select your main house model.
    ii. Repeat until all of the meshes, except for the generator, are within the Blueprint.
        a. Make sure all of the component meshes are children of the Root outpost.
        b. EX:



8. Place BP_House on the Island near Player Start.
    i. While selecting an object and pressing End, the object will drop down to the nearest plane. Use this to avoid the terrain from clipping into the house.

9. Place the outpost_generator static mesh next to the BP_House in the scene.
    i. We will be making changes to outpost_generator that can not be done as a Blueprint Component in future exercises. Make sure that the generator is **not** a component of BP_House.

10. Hit play to preview the building. You should be able to walk through the imported meshes, a problem we will fix later.

Result

There should be a checkered textured house on the island.

Submission

Packaged projects will be checked in class on Monday (9/22/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Official Documentation of the FBX pipeline:
https://docs.unrealengine.com/latest/INT/Engine/Content/FBX/index.html

CGT 345

Due Monday (9/29/2014)

Goal

Create materials and textures for the imported meshes.

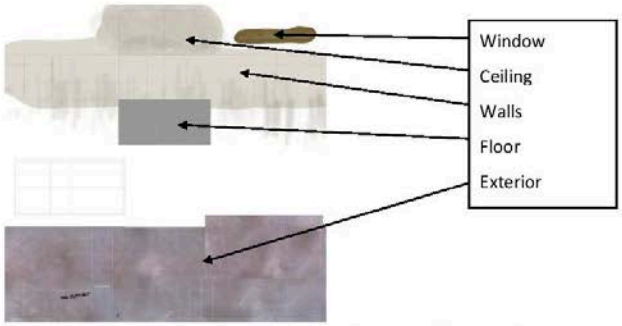
Setup

1. Create a copy of you previous project.

Texture the models

1. Export UV maps of each model
    i. Select the model and open the UV Texture editor. Select the UVs and go to Polygons>UV Snapshot
    ii. Choose an appropriate size for the texture. For larger objects that need more detail like the outpost or generator, you'll want at least 1024x1024. For smaller objects like the matchbox and table, 512x512 is fine.
    iii. Set the filetype to something with transparency (PNG).
    iv. Change the file name to something other than outUV. Naming it after the model is a good idea (ex. outpost_uv, generator_uv, etc...)

2. Texture the objects
    i. Open the UV maps in Photoshop.
    ii. Using the UVs as a guide, paint in textures for the objects. Keep the UV lines on a separate layer at the very top.
    iii. If you would rather use photographic textures, you can use a site like CGTextures.com.
    iv. When you are finished, you will have something that looks like this:



v. Notice that the UV lines are still visible. Uncheck the top layer visibility in Photoshop and that will go away.

vi. Save your file as a JPG in your Maya project's sourceimages folder
vii. Repeat this to make a texture for each model (excluding the generator and the lightbulb). The table and light can be solid colors if you wish, but the rest should have either hand painted or photographic textures.

3. Texture Generator

i. The generator will have 5 different textures that will be swapped in game. A texture will be need for Power Level 0 to Power Level 4.

Materials

1. In the Unreal Editor, import all of your textures for your models in the sourceimages folder into the Textures folder under Game.

2. Create a material for each object in the Materials folder. Each one should be named M_ObjectName.
    i. Attach each texture to their correct material in the Material Editor

3. For outpost_generator, right click on the Texture Sample and select Convert to Parameter. Name the parameter Texture_Sample_2D.
    i. EX:

4. Create a glass window texture using this tutorial:
http://www.youtube.com/watch?v=sRQ5z8i_SV8

5. Copy the glass material and rename it M_lightbulb.
    i. Create a Vector 3 node with the value 1,1,1.
    ii. Multiply this node by 10 and connect the output to Emissive Color.
    iii. Change the first value of the Lerp connected to Opacity, should be a value of 0.008, to .75.

6. Go into BP_House and add lights.
    i. Use a Spot Light to place in the inside light.
    ii. Use a Point Light for the outside light
        a. Make the Point Light a green color for now.
        b. Make sure the light is only showing below your light source.

Result

All objects in the previous exercise are textured with glass and working lights.

Submission

Packaged projects will be checked in class on Monday (9/29/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Material Tutorial:
http://www.youtube.com/watch?v=lngF4VVNER4&list=PLZIv_N0_O1gaCL2XjKluO7N2Pmmw9pvhE&index=98

CGT 345

Due Monday (10/6/2014)

Goal

Create a target range to be used in the next exercise.

Setup

1. Create a copy of you previous project.

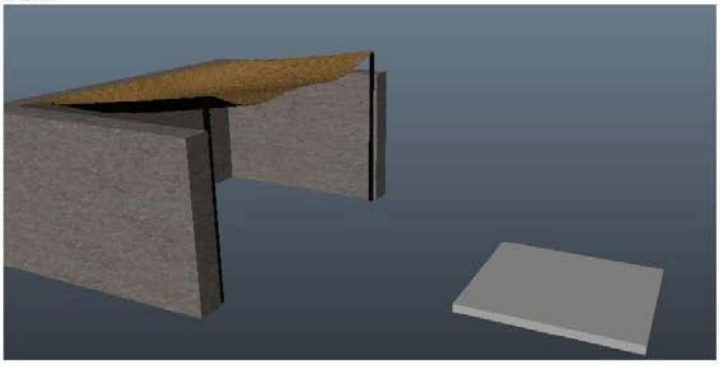
Shooting Range

1. In Maya, build a model of a coconut shy shack.

2. This should be a structure with 3 walls
   i. There should be some sort of mat to stand on at the entrance so the player knows where to stand.
   ii. A ceiling could be added.

3. UV unwrap the model.

4. Create a texture for the shack using photographic textures.
   i. EX:



5. Go to http://www.smart-page.net/smartnormal/ to create a normal map of the textures.

Target

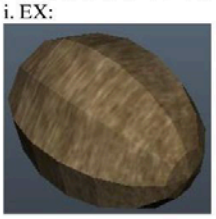
1. Model a target for the player to shoot at.
    i. Should have a base, pole, and a target on top of the pole.
    ii. Make sure that the Pivot is at the base of the pole. Press insert to move the Pivot.

2. Texture the target.
    i. The base and pole can be a solid color, but the target should have some kind of image.
    ii. Make sure the pole and target are separate objects.
    iii. EX:

Coconut

1. Should be little more than an elongated sphere.

2. UV unwrap and texture the coconut.
    i. EX:



Importing

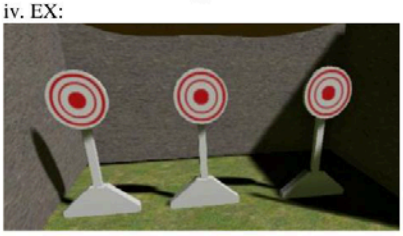
1. Import the shack, target, and coconut into Models folder in the Unreal Editor.
    i. Make sure each is in a fbx format.

ii. Also, have Import Materials checked under Advanced.
iii. Make sure to also import and connect the normal maps to the correct material.

2. Make sure to add collisions to each object.
   i. Double click the static mesh to open up the editor.
   ii. For simple meshes, use a type of Simplified Collision found at the top under Collision.
   iii. For other objects, such as the shack, change the Collision Complexity to Use Complex Collision As Simple.

3. In the Blueprints folder create an Actor blueprint called BP_Shack.
   i. Place the shack meshes as Static Mesh components. Do not make the Mat a component.
   ii. Place BP_Shack in the scene along with the Mat static Mesh.
   iii. Make sure to place these objects away from BatterySpawn.
   iii. EX:

4. Create another blueprint called BP_Target.
   i. Use Static Mesh Components to place the target, or parts.
   ii. Make sure the first component is the pole with the Pivot at the base.
   iii. Place three BP_Target's in the scene under the shack.
   iv. EX:



5. Place a group of Coconut meshes near the mat.
   i. EX:

6. The end result should look something like this:

Result

The scene now has a shack with targets within it. Outside is a mat along with a pile of coconuts.

Submission

Projects will be checked in class on Monday (10/6/2014).

Extra Help:

1. You can also use GIMP and its normal texture plugin to create a normal map:
    i. GIMP: http://www.gimp.org/
    ii. Normal Map Plugin: http://registry.gimp.org/node/7490

2. You can also go here to find textures: http://cgtextures.com/

CGT 345

Due Wednesday (10/15/2014)

Goal

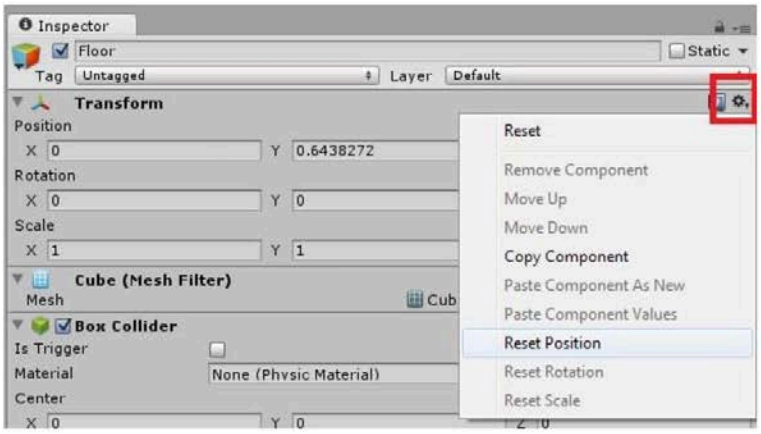
In this exercise, you will create a basic game prototype. You will be exposed to various Unity GameObjects and Components. Perform the following steps.

Setup

1. Create a New Project (File>New Project...)
2. Click Browse... and choose a location on your hard drive.
3. Select the following libraries:
    a. CharacterController
    b. Skyboxes
    c. Terrain Assets
    d. Water

Create a floor

1. From the Hierarchy Window, click Create>Cube
2. Select the Cube listing in the Hierarchy Window and rename it to "Floor".
3. Move the floor to the center of the world. With the Cube selected, go to the Inspector Window and click the Cog icon, then select Reset Position.



4. Turn the cube into a floor. In the Inspector Window, set the Transform Scale to:

   X: 100 Y: 1 Z: 100

Add a light to the scene

1. From the Hierarcy Window, click on the gray background in the window to make sure no other objects are selected.

2. Select Create>Point Light.

3. Move the light above the floor. In the Inspector window, set the Transform Position to:

   X: 0 Y: 20 Z: 0

4. Extend the light's range. In the Inspector window, set the light's Range to 40.

Build a Brick Wall: The Master Brick

We will make a brick wall that will start with one Master Brick that will then be cloned multiple times to make a wall.

1. Build the master brick.

    a. From the Hierarchy window, click on the gray background in the window to make sure no other objects are selected.

    b. Select Create>Cube

    c. Move the cube to the floor

        i. In the Inspector window, set the Transform Position to:

            X: 0 Y: 1 Z: 0

    d. Focus the scene on the cube

        i. Make sure the cube is selected in the Hierarchy window.

        ii. Hover the mouse over the Scene window.

        iii. Press the F key.

    e. Add physics to the cube

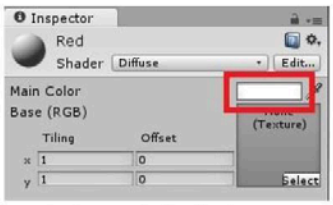
        i. From the Hierarchy window, select the cube

        ii. From the Top menu, select Component>Physics>Rigidbody f. Color the cube

        i. Create a new Material. From the Project window, click Create>Material.

        ii. Name this new Material "Red".

        iii. In the Inspector window, click the Color Picker icon



        iv. Set the color to any shade of red.

v. Apply the Material to the cube. From the Project window, drag the Red Material over to the Hierarchy window and drop it on the cube

2. Test the brick

a. In the Hierarchy Window, select the cube

b. Lift and rotate the cube.

i. Set the Position Y to 15 ii. Set the Rotation X to 40

c. Press the Play button.

d. Verify that the cube falls from the sky and tumbles a little.

e. When done testing, press the Play button again (NOT the Pause button).

f. Set the Y position back to 1 and the X rotation back to 0.

Building a Brick Wall: The rest of the bricks

1. Duplicate the Master Brick

a. In the Hierarchy window, click on the cube

b. Right click and select Duplicate

2. Move the new Cube using Snapping

a. Hold down the Control key.

b. In the Scene window, click and hold on the X Axis handle (Red)

c. Using the mouse move the new cube to the right, right next to the original cube.

3. Repeat step 2 until you have 10 cubes in a row.

4. Create an empty GameObject. From the Top menu select GameObject>Create Empty.

5. Rename the empty GameObject to CubeHolder

6. Move the CubeHolder. In the Inspector window, change the Transform Position:

X: 4.5 Y: 0.5 Z: -1

7. Move all of the cubes to the CubeHolder.

    a. In the Hierarchy window, click on the top cube, hold the shift key, then click on the bottom cube.

    b. Drag the selected cubes and drop them on the CubeHolder.

8. Duplicate the CubeHolder

    a. In the Hierarchy window, select the CubeHolder

    b. Duplicate it.

    c. Hold the Command key. In the Scene window, click and hold on the Y Axis handle (Yellow)

    d. Using the mouse, move the new group of cubes up, on top of the original group. 9.

    Repeat step 8 until the wall is 10 cubes high

Knock down the wall

1. Move the MainCamera

    a. In the Hierarchy window, select the MainCamera

    b. In the Inspector window, change the Transform Position to:

        X: 4 Y: 3 Z: -15

    c. All rotation values should remain at 0

2. Create a C# Script

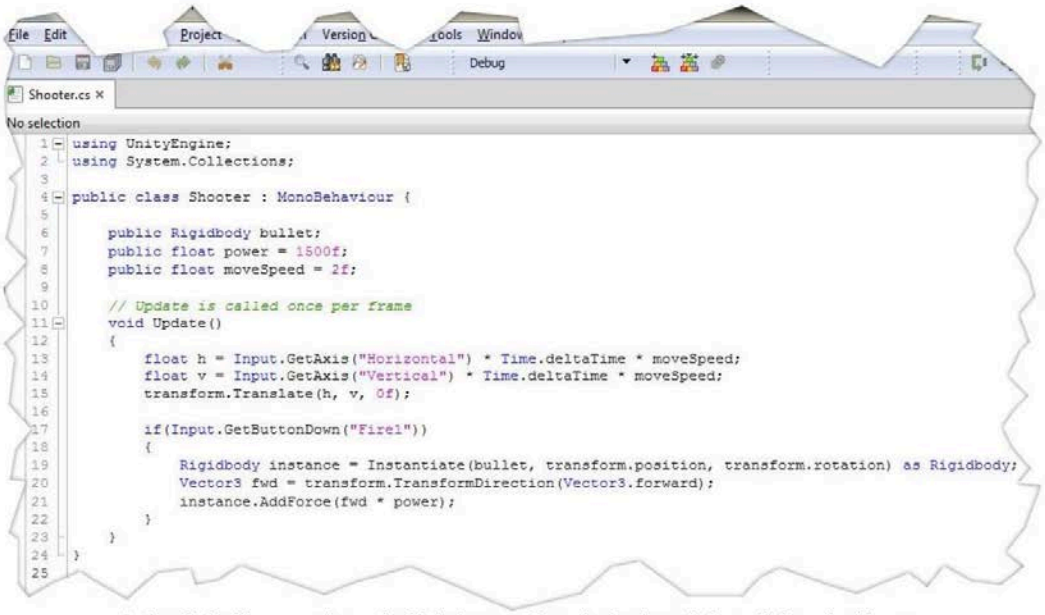
    a. In the Project window, select Create>C# Script

    b. Name the Script "Shooter"

    c. In the Project window, double-click the Shooter script. A new window will open in MonoDevelop.

    d. Replace the code that is there with the following code. Spacing and capitalization matters!

    (Note: In MonoDevelop I find it helpful to go to Tools>Options>Text Exitor>General and uncheck Commit completion list selection with space or punctuation)



3. Attach the Shooter script to the MainCamera. From the Project window, click on the Shooter script and drag it to the Hierarchy window where you will dorp it on the MainCamera object.

4. Save the Scene.

    a. From the top menu, select File>Save Scene As...

    b. Name the scene "Ex_01"

    c. Make sure the scene is saved in the Assets folder (It is the default choice).

5. Create the bullet

    a. From the Hierarchy window, select Create> Sphere

    b. Create a Material for the sphere.

        i. We will first make a folder to hold our materials.

        ii. From the Project window, select Create>Folder

        iii. Name the folder "Materials"

        iv. Drag and drop the Red Material we made earlier into the Materials folder we just created

v. Select the new Materials folder.

vi. Create a new material and name it "BulletColor". Set it to a shade of blue.

c. Apply the Material to the Sphere

6. Make the bullet have physics properties. Select the sphere and add a Rigidbody component.

7. Convert the Sphere to a Prefab.

a. In the Project window, create a Prefabs folder

b. Drag the sphere from the Hierarchy window and drop it in the Prefabs folder.

c. In the Project folder, rename the sphere object to Projectile.

d. In the Hierarchy window, delete the sphere object (Right click>Delete)

8. Assign the bullet to the Projectile Prefab.

a. In the hierarchy window, select the Main Camera.

b. In the Project window, drag the projectile prefab to the Inspector window and dorp it on the Shooter:Bullet value. Proior to dropping it will say "Bullet None (Rigidbody)"

c. It should now read "Bullet Projectile (Rigidbody)"

d. Save the scene

9. Verify that it works

a. Click Play

b. Enjoy!

c. Once you've played with it for a bit, try changing it around some and experimenting.

Submission

Packaged projects will be checked in class on Wednesday (10/15/2014).

Building a standalone exe

Go to File > Build Settings in Unity. Make sure all of the scenes you want to use are included in the "Scenes in Build" window. If you have no script built into your game to transition from one scene to the other, only the first scene will be available when you run the exe. Your build must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include the data folder with your exe or it will not run.

CGT 345

Due Wednesday (10/15/2014)

Goal

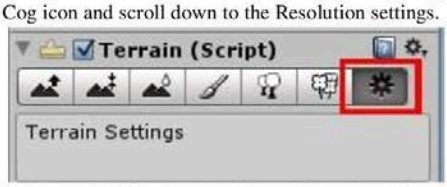
In this exercise, you will create your own private island.

Setup

1. Create a New Project (File>New Project...)
2. Click Browse... and choose a location on your hard drive.
3. Select the following libraries:
    a. CharacterController
    b. Skyboxes
    c. Terrain Assets
    d. Water (Basic)

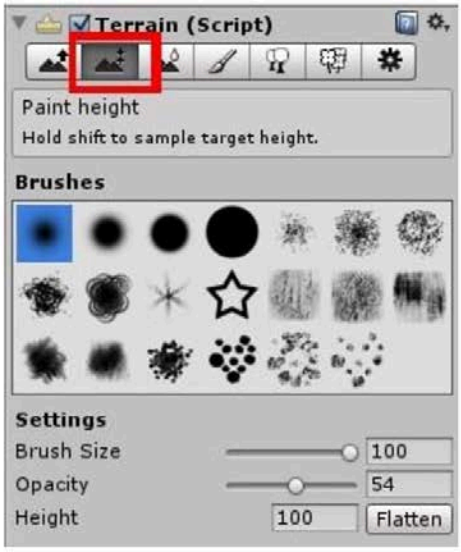
Create the terrain

1. Create the geometry
    a. From the top menu, select Terrain > Create Terrain
    b. Select the Terrain in the Hierarchy and go to the Inspector window. Click on the Cog icon and scroll down to the Resolution settings.

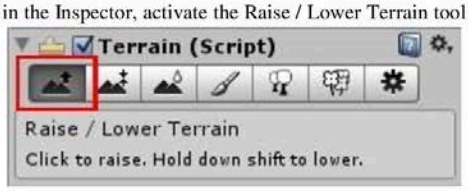


    c. Set Width = 500, Height = 600, and Length = 500.

    d. Select the Paint Height tool. Set Height = 30 and click Flatten.

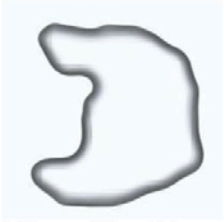


2. Carve the shore
    a. In the Hierarchy, select the Terrain.
    b. in the Inspector, activate the Raise / Lower Terrain tool by clicking its icon.



    c. Set Brush Size = 100, Opacity = 75
    d. Change the Scene view to the top view by clicking the green Y arrow in the view gizmo:

e. Zoom in or out (Alt + RMB or use Scroll Wheel) until you can see the whole terrain as a square on the screen
f. While holding the Shift key and the left mouse button, draw near the edges of the terrain to carve out a shoreline. Use the first brush. Try to match the picture below as closely as possible



g. Add some hills and depressions on the island using the Raise Height tool. Be creative. Make sure to leave lots of flat space available for other items to be added later. Some tips:
    i. To make a dip, hold the shift key while clicking.
    ii. To make a hill, just click where you want the hill placed. iii. Experiment with the different brushes to get different hill shapes. iv. Use the Smooth Height tool to round off some of the sharp edges.
    v. Make sure you don't cover the land completely with hills. You will want a flat path across the island.

Volcano!

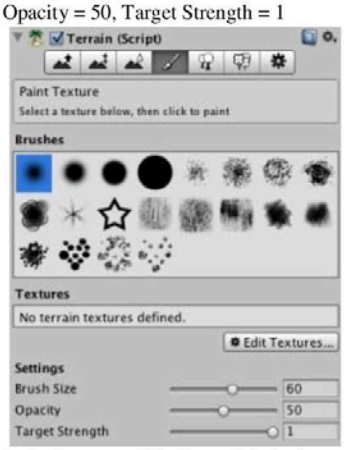
1. Activate the Paint Height tool. Use the same brush you have been using (the first one).
2. Set the Paint Height tool's parameters to Brush Size = 75, Opacity = 50, Height = 100
3. In the scene window, Draw a circle in the lower left corner of the island. It should look roughly like this:

4. Draw the mouth of the volcano. Again, using the Paint Height tool, change the settings to: Brush Size = 30, Opacity = 50, Height = 20. Now draw in the mouth of the volcano. It should appear as follows:

5. Activate the Smooth Height tool. Set its attributes to: Brush Size = 30, Opacity = 100.
6. Smooth out the rim of the volcano's mouth by using your mouse to draw around the upper lip of the volcano.
7. Remember, if you lose track of your terrain in the Scene window, press the F key while the mouse is over the window.

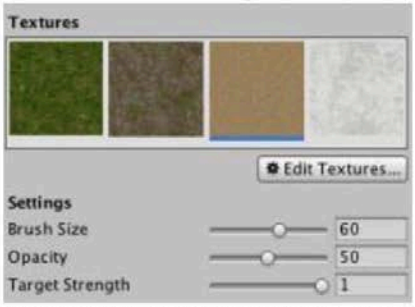
Texturing

1. Activate the Paint Textures tool and set its parameters as follows: Brush Size = 60, Opacity = 50, Target Strength = 1

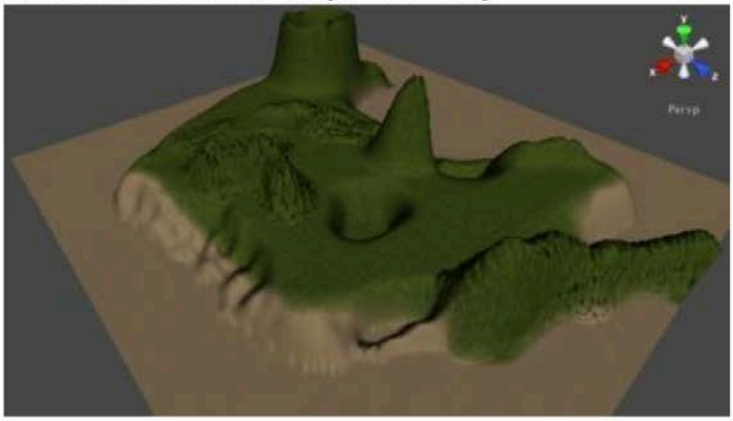


2. In the Inspector Window, click the button called "Edit Textures" then click "Add Texture".
3. Click the circle to the right of the Splat field. This will show all of the available textures. Select the one called "Grass (Hill". Now click the Add button.

4. Repeat steps 2 and 3 for 3 more textures: Grass & Rock, GoodDirt, and Cliff(Layered Rock). For the cliff, set the Tile Size X = 70 and Tile Size Y = 70.
5. Notice that the terrain is covered in the Grass texture. This is because it was the first Texture you added. The rest of the textures you will apply by painting with the mouse.
6. Select the texture called GoodDirt (See below, the selected texture has a blue line under it). Set the brush Size = 60, Opacity = 50, Target Strength = 1

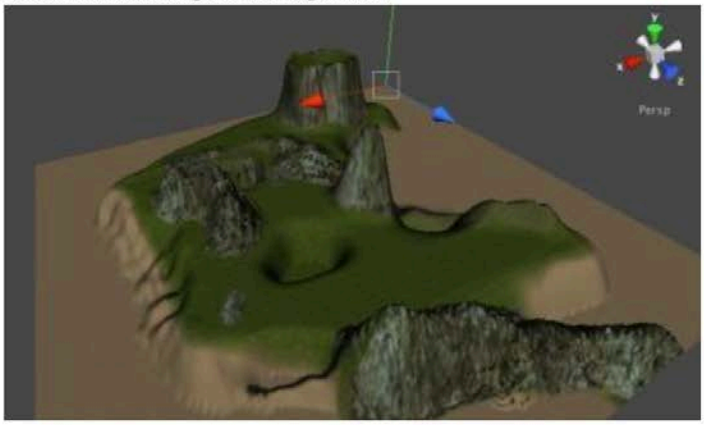


7. Using the mouse, paint beaches along the coast of your island.
    a. Remember you can switch back and forth between the Top and Perspective views using the View Gizmo.
    b. In the perspective view, Alt clicking will rotate the scene.
    c. Mouse wheel can be used to zoom the scene in and out.
    d. When finished, it should look something like the following:



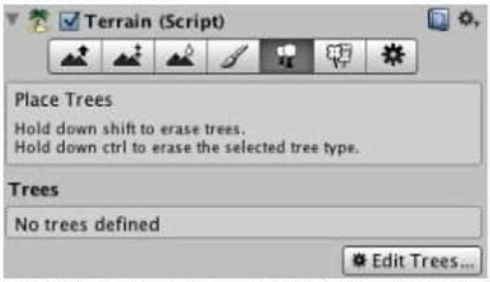
8. Switch the active texture to Grass & Rock; Brush Size = 25, Opacity = 30, Target Strength = .5
9. Use the Grass & Rock Texture to paint any low hills you have created.

10. Lastly, the volcano and Rock outcrops need texturing. Select the Cliff (Layered Rock) texture. Set Brush Size = 20, Opacity = 100, Target Strength = 1
11. Using the Cliff(Layered Rock) Texture, paint the inner and outer walls of the volcano. Also paint any protruding rocks you have created in your scene. WHen you are finished, it should look something like the image below:



Trees

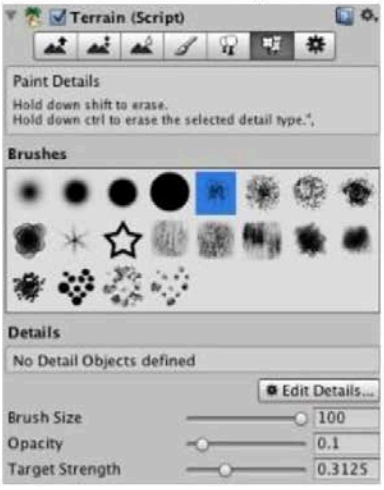
1. Activate the Tree tool:



2. Click the "Edit Trees" Button. Click "Add Tree". Click the circle to the right of the Tree field. Double click "Palm". Now set the bend factor to 2.
3. Set the following settings for teh Tree tool, Brush Size = 15, Tree Density = 40, Color Variation = .4, Tree Height = 50, Tree Width = 50, Variation = 30.
4. Single click locations around the island to place trees. If you make a mistake, hold Shift and click to erase trees.

Grass

1. Select the Paint Details tool. Now set brush size = 100, Opacity = .1, Target Strength = 0.3125. Note: You will be using a different brush than you have previously, see below:



2. Click "edit grass", Click "Add Grass Texture". Click the circle to the right of the field called "Detail Texture". Select the icon called "Grass". Keep the rest of the values at their default values.
3. Single click points on your island to apply grass.
    a. Do not overdo it, too much grass will slow the game down tremendously.
    b. If you have zoomed out too far, you cannot see the grass.

Here comes the sun

1. From the TOp Menu, select GAmeObject > Create Other> Directional Light.
2. In the Inspector window, set Position X = 0, Position Y = 250, Position Z = -200, Rotation X = 15, Rotation Y = 0, Rotation Z = 0.
3. From the Toip Menu, select Assets > Import Package > Light Flares. Now click "Import".
4. In the inspector, click the circle to the right of the field called "Flare", double click "Sun"
5. From the Top Menu, select Edit > Render Settings.
6. In the Inspector Window, click the circle to the right of the field called "Skybox Material". Double click the icon named "Sunny 2 Skybox".

Water

1. In the project window, navigate to Standard Assets > Water (Basic). Drag "Daylight Simple Basic" over to the Hierarchy Window; it should not be dropped on any other object.
2. In the inspector Window, set the Position X = 250, Position Y = 4, Position Z = 250, Scale X = 1600, Scale Y = 1, Scale Z = 1600.

Walk Around

1. Zoom in and locate the place on the island where you want to begin walking.
2. In the Project Window, navigate to Standard Assets > Character Controllers. Select "First Person Controller", drag it to the Scene Window, and drop it at the location you identified.
3. Using the Axis Arrows attached on the First Person Controller object in the scene, move the capsule up so that it is above the ground.
    a. If it is too far above, you could fall and get injured, if it is too far below you will fall through the terrain.
4. In the hierarchy window, delete the old MainCamera object.
5. Click the Play button. Using the keyboard, you can walk around the island you have created (using the WASD keys and Spacebar). Use the mouse to look around. Enjoy!

Submission

Packaged projects will be checked in class on Wednesday (10/15/2014).

Building a standalone exe

Go to File > Build Settings in Unity. Make sure all of the scenes you want to use are included in the "Scenes in Build" window. If you have no script built into your game to transition from one scene to the other, only the first scene will be available when you run the exe. Your build must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include the data folder with your exe or it will not run.

CGT 345

Due Monday (10/20/14)

Goal

In the exercise, you will be adding meshes and animations to your first person controller.

Setup

1. Create a copy of you previous project.
    i. Either Clone
        a. Open Unreal Engine.
        b. Go to Library.
        c. Under My Projects, click the arrow next to Open beneath your project file from the previous week.
        d. Click on Clone
        e. In Name, make sure it keeps the original project name. If it has a different name, you will be unable to recompile in the editor due to different .dll and .sln files from the project name.
        f. Click Create.
        g. Once the clone appears, click the Open tab beneath the project.
    ii. Or copy the folder to have a backup version.
    iii. Make sure the project name is the same. If it is different from the .sln or .dll files then you will be unable to recompile code in the editor.

Adding a Mesh to your Character

1. Follow the directions from the Adding a Mesh to you Character to Adding a First Person Mesh Tutorial, stop before Adding Projectiles and Shooting:
https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial#Adding_a_mesh_to_your_Character

2. If you are unable to download the mesh from the links then import the fbx files provided.

Animation

1. Follow the instructions for adding animation to your character:
https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial#Animating_Your_Character

2. If you are unable to download the animation sequences from the link then import the fbx files provided.

3. This section features images that are out of date. Make sure to read the text when using the animation blueprint.

Result

The characters arms should go through a run, idle, and jumping state.

Submission

Packaged projects will be checked in class on Monday (10/20/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Official Documentation of Animation Blueprint:
https://docs.unrealengine.com/latest/INT/Engine/Animation/AnimBlueprints/index.html

CGT 345

Due Monday (10/20/14)

Goal

Create a working FPS controller with a mesh and animation.

Setup

1. Create a copy of you previous project.
    i. Either Clone
        a. Open Unreal Engine.
        b. Go to Library.
        c. Under My Projects, click the arrow next to Open beneath your project file from the previous week.
        d. Click on Clone
        e. In Name, make sure it keeps the original project name. If it has a different name, you will be unable to recompile in the editor due to different .dll and .sln files from the project name.
        f. Click Create.
        g. Once the clone appears, click the Open tab beneath the project.
    ii. Or copy the folder to have a backup version.
    iii. Make sure the project name is the same. If it is different from the .sln or .dll files then you will be unable to recompile code in the editor.

Blueprints

1. If you haven't already, create a Blueprints folder.
    i. Inside this folder, place a Blueprint with a custom class of HUD called BP_FPSHUD.
    ii. Place another Blueprint with the Character class called BP_FPSCharacter.
    iii. Place a Game Mode Blueprint called BP_FPSGameMode.

2. Open BP_FPSGameMode.
    i. In the Defaults tab, set Default Pawn Class to BP_FPSCharacter.
    ii. Set HUD Class to BP_FPSHUD.
    iii. Compile and Save.

3. Save and exit the editor.

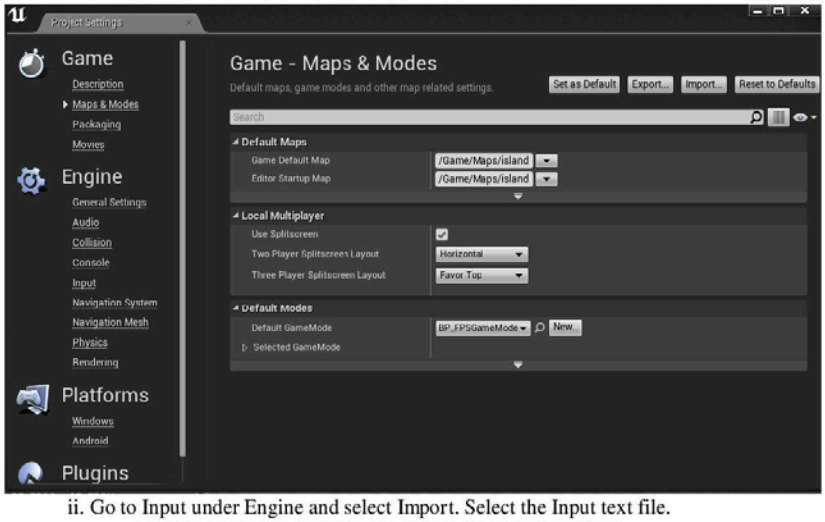
Project Settings

1. Get the Game and Input text files off of Blackboard.

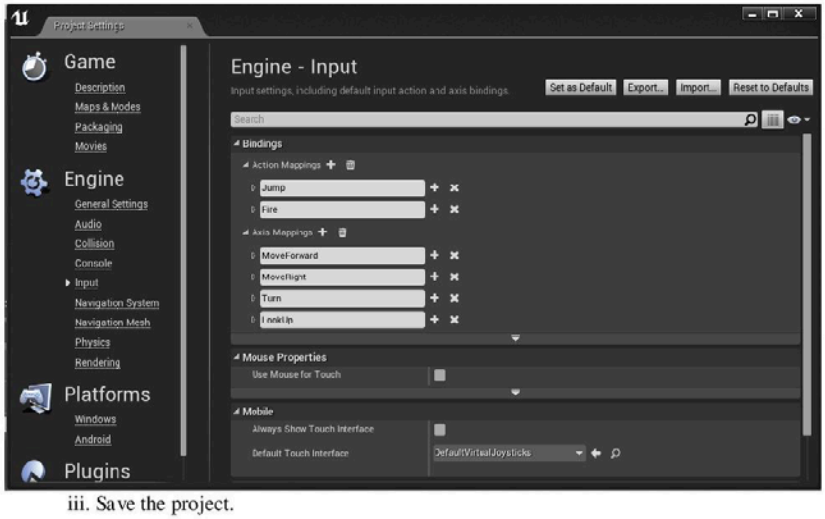
2. Open you project folder, go to Saved > Config > Windows.
    i. Replace or paste the content from the two text files into the Game and Input within the Windows folder.
    ii. You may need to change the Map settings depending on the file location and name of the file. By default, it is set to look for an island file within Maps.

3. Open up your project again in the Unreal Editor.

4. Go to Edit > Project Settings.
    i. Go to Maps & Modes and select the Import button. Select the Game text file.
        a. The Default Maps settings and Default Modes should change.
        b. EX:



    ii. Go to Input under Engine and select Import. Select the Input text file.
        a. Inputs should be added to the Action and Axis Mappings.
        b. EX:



iii. Save the project.

FPS Character

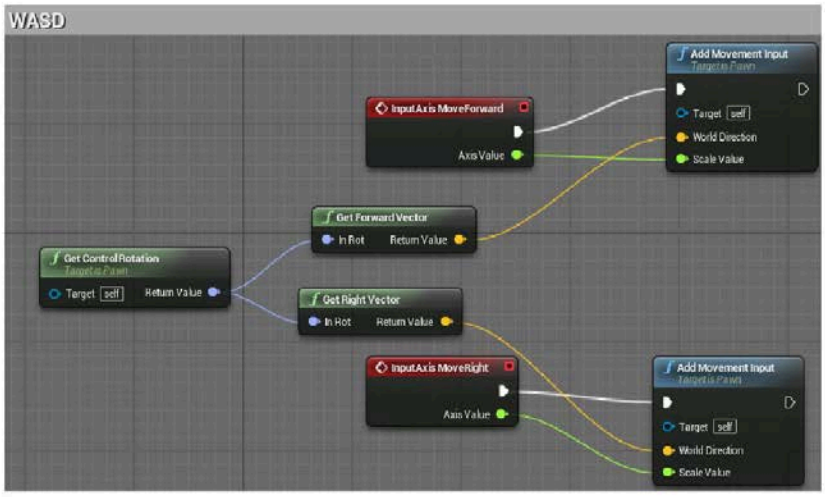
1. Open BP_FPSCharacter and go to the Graph tab.

2. Right click and place the Axis Event of MoveForward.
    i. Drag the Output arrow and place Add Movement Input.
    ii. Drag Axis value into Scale Value.

3. Right click and place the Axis Event of MoveRight.
    i. Drag the Output arrow and place Add Movement Input.
    ii. Drag Axis value into Scale Value.

4. In Find a Node, search and place Get Control Rotation.
    i. Drag the Return Value and place Get Forward Vector.
        a. Drag its Return Value into the World Direction of Add Movement Input of MoveForward.
    ii. Drag the Return Value and place Get Right Vector.
        a. Drag its Return Value into the World Direction of Add Movement Input of MoveRight.

5. EX:



6. Right click and place the Action Event of Jump.
    i. Drag Pressed and place the function Jump.

7. Right Click and place the Axis Event of LookUp.
    i. Drag the Output arrow and place Add Controller Pitch Input.
    ii. Drag Axis Value into Val.

8. Right Click and place the Axis Event of Turn.
    i. Drag the Output arrow and place Add Controller Yaw Input.
    ii. Drag Axis Value into Val.

9. Compile and Save. In Play, you will now be able to move the controller with WASD and the mouse.

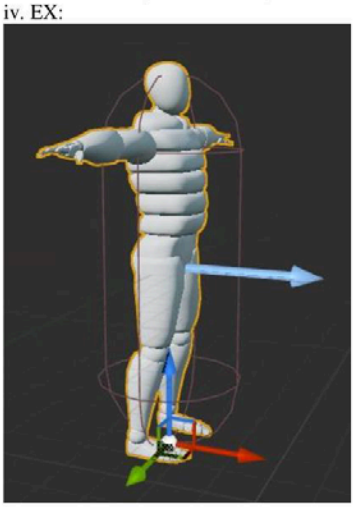
10. You can change the Jump Z Velocity in the Defaults tab of BP_FPSCharacter under Character Movement.

Mesh

1. Import the GenericMale FBX from Blackboard. Under Advanced, make sure Import Material is checked.

2. Open BP_FPSCharacter and go to the Components tab.
    i. Select Mesh from Components.
    ii. Go to Mesh under Details and select the GenericMale skeletal mesh.

iii. In Transform, under Details, set the Z Location to -88.
iv. EX:



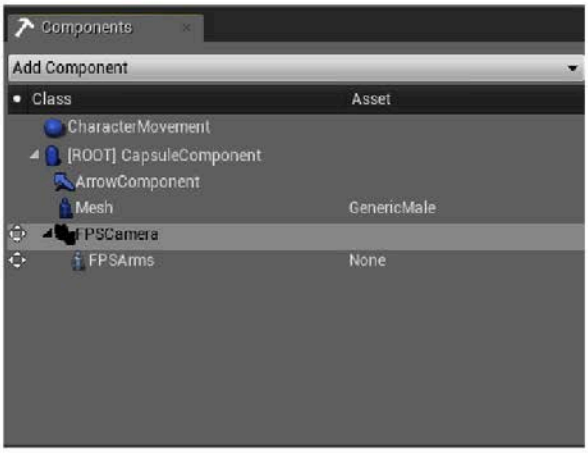
3. Compile and Save.

4. In Play, press Shift + F1 and select Eject. You should see the generic male mesh.

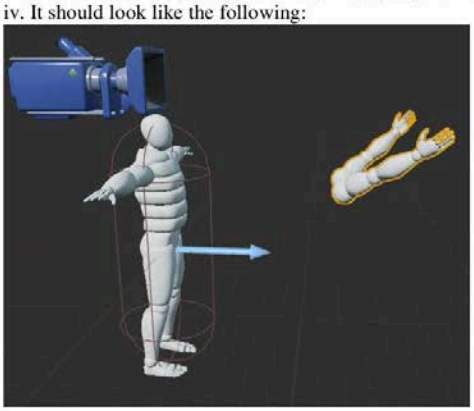
5. Go back to the Components tab of BP_FPSCharacter.
    i. Under Details of Mesh, go to Rendering.
    ii. Select the arrow to expand and check Owner No See.

6. Import the HeroFPP FBX from Blackboard. Under Advanced, make sure Import Material is checked.

7. Add a Camera component called FPSCamera.
    i. Set the Location Z of FPSCamera to 114.
    ii. Add a Skeletal Mesh component called FPSArms.
    iii. Drag FPSArms over FPSCamera to attach.
    iii. EX:



8. Select FPSArms.
    i. Under Mesh, select HeroFPP from the drop down menu.
    ii. Under Rendering, select the expansion arrow and check Only Owner See.
    iii. Under Transform, set Location to [240, 0, 35] and Rotation to [-180, 50, -180].
    iv. It should look like the following:



9. Compile and Save. In play, the arms should be visible to the camera.

Animation

1. Follow the instructions for adding animation to your character:
https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial#Animating_Your_Character

2. If you are unable to download the animation sequences from the link then import the fbx files provided.

3. This section features images that are out of date. Make sure to read the text when using the animation blueprint.

Result

There should be a working character controller with an animating mesh.

Submission

Packaged projects will be checked in class on Monday (10/20/14).

Building a standalone exe

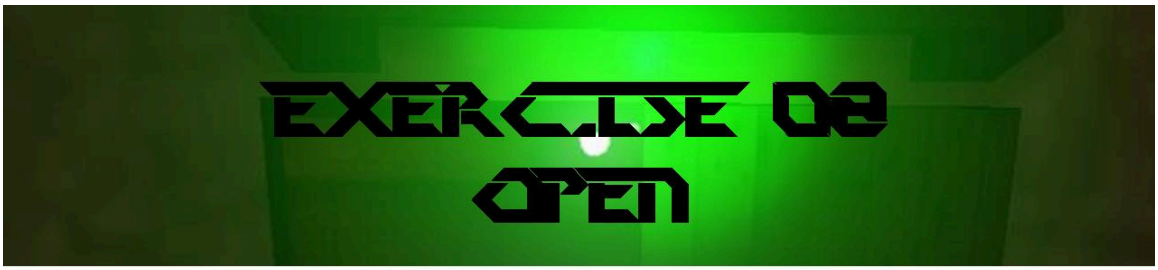
Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Official Documentation of Animation Blueprint:
https://docs.unrealengine.com/latest/INT/Engine/Animation/AnimBlueprints/index.html

2. A C++ tutorial series:
http://www.youtube.com/watch?v=vtcWrcscXos&list=PLZIv_N0_O1gaCL2XjKluO7N2Pmmw9pvhE&index=2



CGT 345

Due Monday (10/27/14)

Goal

Create collision areas for the outpost and animate the door.

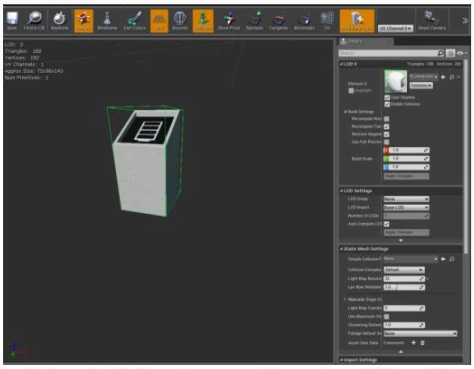
Setup

1. Create a copy of you previous project.

Collisions

1. In the Models folder, open the house model.
    i. In Details under Static Mesh Settings, Change the Collision Complexity to Use Complex Collision As Simple.
    ii. Save and the controller should be unable to go into the house.

2. Open the other models to edit.
    i. The rest of the models can use a simpler collision than the house.
        a. In the mesh editor, go to the top and select Collisions > 6DOP Simplified Collision.



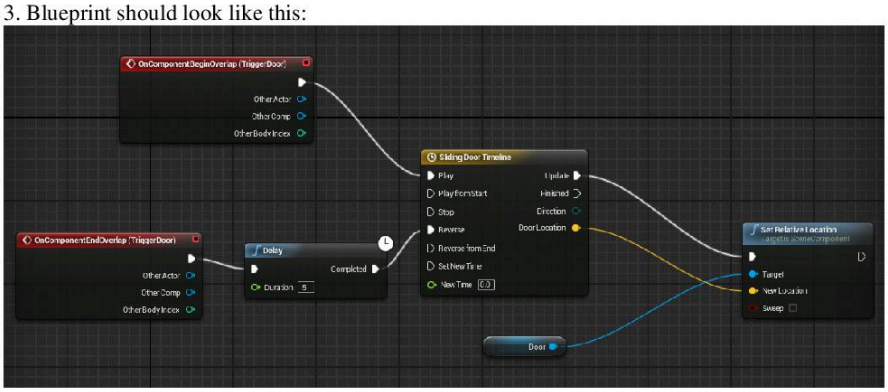
        b. Select Collisions between Bounds and Show Pivot to preview the collision volume.
        c. Save and repeat for each mesh.

Open Sesame

1. Follow this tutorial:
http://www.youtube.com/watch?v=URG3YNsUCQ8&index=65&list=PLZlv_N0_O1gaCL2XjKluO7N2Pmmw9pvhE
    i. Instead of creating the trigger the in scene, go to BP_House and Add Component then Box under Shapes.
    ii. Use BP_House instead of Level Blueprint.

2. Go on to Part 2: http://www.youtube.com/watch?v=d2Ua6ncLcVY
    i. Ctrl drag the door model in My Blueprint in the scene instead of the reference.
    ii. Connect door to Set Relative Location.
    iii. Ignore calculating the door's initial location.
    iv. The error of Begin and End Overlap no longer occurs in the updated versions of the engine. Still follow the custom collision directions.

3. Blueprint should look like this:



Result

The character controller should not be able to walk through any part of the outpost. The position of the character should trigger the door to open or close.

Submission

Packaged projects will be checked in class on Monday (10/27/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not

the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. More blueprint tutorials: https://www.unrealengine.com/blog/new-blueprint-tutorials

CGT 345

Due Monday (11/3/14)

Goal

Have the player collect batteries in order to open the outpost's door.

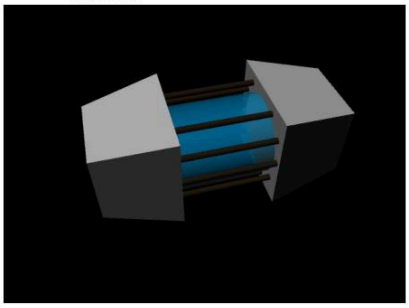
Setup

1. Create a copy of you previous project.

2. Create a Sounds folder and import the sound files.
    i. You can also find your own here: http://www.freesound.org/
    ii. Make sure you have a Door Lock, Door Open, and Collect sound.

Power Cell

1. Model a power cell.
    i. It should roughly look like a battery, fuel, or something that can power a generator.
    ii. Can UV unwrap and texture but solid colors are okay.
    iii. EX:

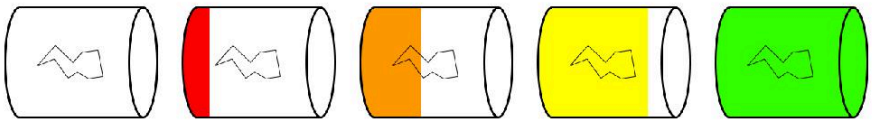


2. Import the mesh into the Models folder in the editor.
    i. Make sure that Import Material is checked under Advance.

HUD Elements

1. Create a heads up display that will show the amount of power cells collected.

i. Keep the dimensions a power of 2 (256x256 or 512x512).
ii. Make a 5 different power levels to match the generator.
iii. EX:



2. Save them as a PNG file type, or another file type that supports transparency.
i. When saving, make sure they have names that are easy to reference later. (power0, power1…power4)

3. Import the textures into the editor within the Textures folder.

Sound

1. Open your Sounds folder.
i. Right Click doorlocked and select Create Cue.
a. Double Click to open.
b. Disconnect the two nodes.
c. Drag the Output of Wave Player and place Modulator.
A. Set Pitch Min to .7 and Pitch Max to 1.25.
d. Drag the Output of Modulator into Output.
e. Save.
ii. Right Click slidingdoor and select Create Cue.
a. Double Click to open.
b. Disconnect the two nodes.
c. Drag the Output of Wave Player and place Modulator.
A. Set Pitch Min to 1.5 and Pitch Max to 2.0.
d. Drag the Output of Modulator into Output.
e. Save.

Generator

1. Open the material for the generator.
i. Right Click the texture and select Convert to Parameter.
ii. Name it Texture_Sample_2D.

2. Compile and Save.

Pick Up

1. Open the editor and go *File > Add Code to Project...*
i. Choose an Actor as the parent class.

ii. Name it PickUp.

2. Open your code editor and add the follow to your PickUp.h file underneath the GENERATED_UCLASS_BODY():

```
	/** Simple collision primitive to use as the root component */
	UPROPERTY(VisibleDefaultsOnly, BlueprintReadOnly, Category = Pickup)
	TSubobjectPtr<USphereComponent> BaseCollisionComponent;

	/** StaticMeshComponent to represent the pickup in the level */
	UPROPERTY(VisibleDefaultsOnly, BlueprintReadOnly, Category = Pickup)
	TSubobjectPtr<UStaticMeshComponent> PickupMesh;

	/** Variable for rotation per second*/
	UPROPERTY(EditDefaultsOnly, Category = Pickup)
	float RotationRate;

	/**Override the tick function to update every tick*/
	virtual void Tick(float DeltaTime) OVERRIDE;
```

3. Go to PickUp.cpp and add the following to the constructor:

```
	//Object can tick
	PrimaryActorTick.bCanEverTick = true;

	// Create the root SphereComponent to handle the pickup's collision
	BaseCollisionComponent = PCIP.CreateDefaultSubobject<USphereComponent>(this,
TEXT("BaseSphereComponent"));

	//Set sphere component radius, may need a different value than 60
	BaseCollisionComponent->SetSphereRadius(60.0f, false);

	//Set the SphereComponent as the root component
	RootComponent = BaseCollisionComponent;

	//Create static mesh component
	PickupMesh = PCIP.CreateDefaultSubobject<UStaticMeshComponent>(this,
TEXT("PickupMesh"));

	//Attach the StaticMeshComponent to the root component
	PickupMesh->AttachTo(RootComponent);

	//Rotation rate per second
	RotationRate = 180.0f;
```

4. Add the following code below the constructor to rotate the battery:

```
//Set battery rotation
void APickUp::Tick(float DeltaTime)
{
	//Calls all other Tick events
	Super::Tick(DeltaTime);

	//Sets rotation for battery
	FRotator MyRot = GetActorRotation();
	MyRot.Yaw += RotationRate* DeltaTime;
```

```
        SetActorRotation(MyRot);
}
```

5. Build the solution.

6. Open the editor and go to the Blueprints folder. Select New > Miscellaneous > Blueprint Interface. Name it BP_Check.

7. Open BP_Check.
    i. Select the + Function button. Name the new function ShowHint.
        a. In the details of ShowHint, Press New within the Inputs section and name the Boolean variable bIsShowing.
    ii. Select the + Function and name the new function ShowInventory.
        a. Add an Input variable called bIsCounting.
    iii. Select the + Function again and name the new function AddCharge.
        a. Add an Integer for Inputs called PowerUp.
    iv. Add a Function Called DoorStatus.
        a. Make a Boolean input called bIsOpen.

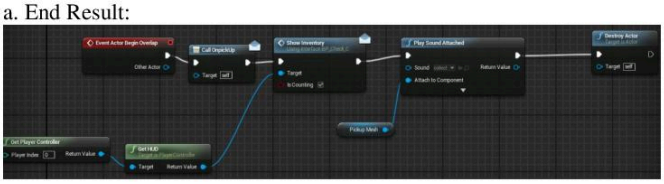
8. Compile and Save BP_Check.

9. Back in the Blueprints folder, create a new Blueprint using the PickUp class as a parent class called BP_Battery.

10. Open BP_Battery and go to the Components section.
    i. Select [ROOT] BaseCollisionComponent.
        a. In Details under Collision, Change the Collision Presets to Custom and Ignore all collisions except for Pawn, which should have Overlap checked.
    ii. Select PickupMesh.
        a. In the Static Mesh section select the Battery that was imported.
        b. Use the same collision settings as the previous step.

11. Go to the Graph section.
    i. In the My Blueprints tab, click on Event Dispatcher. Name the event OnPickUp.
        a. It should be to the right of Macro and Graph.
    ii. Right Click and search overlap. Select the event Event Actor Begin Overlap.
    iii. Drag the arrow and find Call OnPickUp.
    iv. Drag the arrow and place Show Inventory Interface Message.
        a. Make sure that Is Counting is checked within Show Inventory.
        b. Right Click and place Get Player Controller.
        c. Drag the Return Value and place Get HUD.
        d. Drag the Return Value of Get HUD into the Target.
    v. Drag the Output of Show Inventory and place Play Sound Attached.
        a. Set Sound to collect.
        b. Right Click and place a Get of PickupMesh.
        c. Connect to Attach to Component.
    vi. Drag the Output of Play Sound Attached and place Destroy Actor.

a. End Result:



12. Compile and Save BP_Battery.

Generator Instance

1. Open the Level Blueprint.

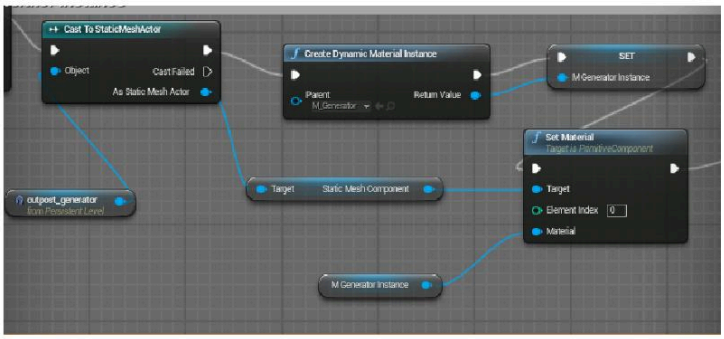
2. Create a new variable with the type actor.
    ii. Place a Get of it on the scene.
    iii. Drag and place Cast To StaticMeshActor.
        a. You can delete the actor variable as we will not be using it.
    iv. Drag the input arrow of Cast To StaticMeshActor into the output of the last Execute Console Command Node.

3. Go back to the scene and select you Generator mesh. In the Level Blueprint, Right Click and add a reference to the mesh.
    i. Connect it to the Object of Cast To StaticMeshActor.

4. The top Output of Cast To StaticMeshActor and place Create Dynamic Material Instance.
    i. For the Parent of Create Dynamic Material Instance, choose M_Generator.
    ii. For the Cast Failed output, you can connect to Print String to check if the code is working.
5. Right Click the Return Value of Create Dynamic Material Instance and select Promote to Variable. Name the new variable M_GeneratorInstance.

6. Search in Find a Node and place a Set Material Node.
    i. Connect the Set of M_GeneratorInstance into the top input of Set Material.
    ii. Drag the As Static Mesh Actor component of Cast To StaticMeshActor into the Target of Set Material.
    iii. Place a Get of the M_GeneratorInstance variable and connect it to Material.

7. This will create an instance of the Generator material that can have its texture changed during runtime.



Spawning

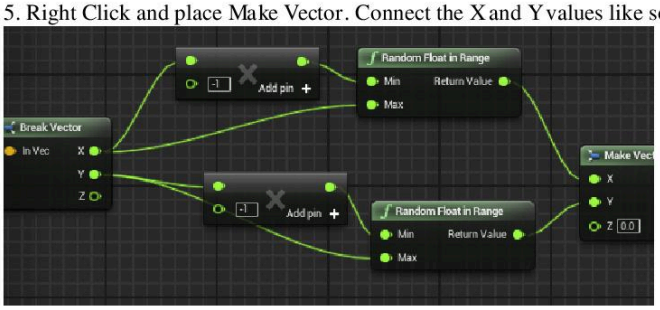
1. Create a trigger volume in the scene and name it BatterySpawn.
    a. The box trigger can be found in the Modes tab under Place > Basic.

2. Scale BatterySpawn in the X and Y direction so that it covers a large area.
    a. Make sure the House or an unreachable area are not within the volume.

3. Make sure that BatterySpawn is selected and go back to the Level Blueprint. Right Click and place a reference of the object.
    i. Drag the Output and place Get Actor Bounds.
    ii. Drag Box Extent and place Break Vector.

4. Right Click and place Random Float in Range.
    i. Place Float * Float and connect it to the Min.
        a. Make one of the values is -1.
    ii. Connect the X value of Break Vector into the empty spot, the spot that isn't -1 of Float * Float, and into the Max of Random Float in Range.
    iii. Do the same for Y.

5. Right Click and place Make Vector. Connect the X and Y values like so:



6. Create another reference to BatterySpawn and drag its Output to place Get Actor Location.
    i. Drag Return Value of Get Actor Location and place Vector + Vector.
    ii. Drag the Return Value of Make Vector into the open spot of Vector + Vector.

7. Place Make Transform. Connect the Output of Vector + Vector into Location.

8. Go back up to Set Material and drag its Output to place a ForLoop.
    i. Set the First Index to 0 and Last Index to 3.

9. Drag Loop Body and place Spawn Actor from Class.
    i. Set Class to BP_Battery.
    ii. Connect the Return Value of the Make Transform node into Spawn Transform.

10. Create a new variable using the Variable Type BP_Battery_C.
    i. Drag a Get of it onto the graph.
    ii. Drag the Output and place Bind Event to OnPickUp.
    iii. Delete the variable.

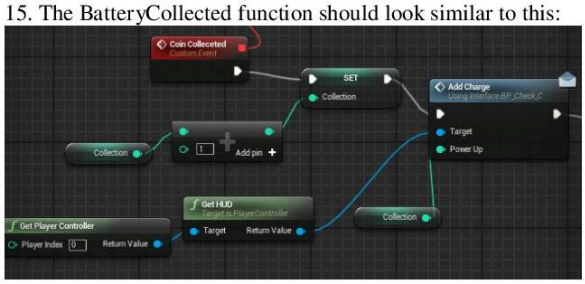
11. Connect the Output and Return Value of SpawnActor BP_Battery_C, formerly Spawn Actor from Class, into the Input and Target of Bind Event to OnPickUp.

12. Create a new Custom Event and name it BatteryCollected.
    i. Place BatteryCollected and connect the red box to Event of Bind Event to OnPickUp.

13. Create a new Integer called Collect. Place a Get and Set of the variable onto the graph.
    i. Drag the Output of Get and place Integer + Integer. Make the value of the open slot 1.
    ii. Connect the Output to the Collect of Set.
    iii. Connect the Output of BatteryCollected into the Input of Set Collect.

14. Drag the Output of Set Collection and place the Interface Messages AddCharge.
    i. Right Click and place Get Player Controller. Drag the Return Value and place Get HUD. Drag its Return Value to Target.
    ii. Place a Get of Collect and connect it to Power Up.

15. The BatteryCollected function should look similar to this:



16. Compile the Level Blueprint. On Play, you should see 4 batteries spawn and then disappear when the character walks through them.

Close and Open

1. Open the BP_House blueprint.

2. Go to Graph and select Blueprint Props. Under Interfaces, Add BP_Check_C.

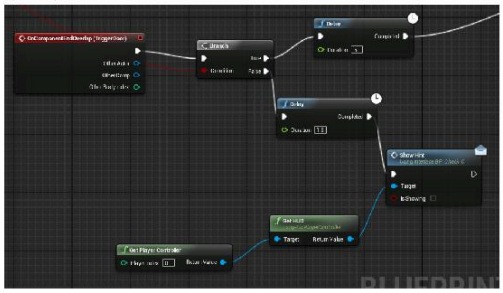
3. Disconnect Begin Overlap and End Overlap event nodes. Move them to the left and Right Click between them and place Event Door Status.
    i. Make sure Event Door Status is in a good spot. We will be using the Is Open component frequently for this graph.

4. Go down to OnComponentEndOverlap and drag the top Output to place a Branch.
    i. Drag the Is Open of Event Door Status into the Condition of the Branch.

5. Drag False and place a Delay node with a Duration of 1.5.

6. Drag the Completed of Delay to place the Interface Messages of ShowHint.
    i. Right Click and place Get Player Controller. Drag Return Value and place Get HUD. Connect the Return Value to Target.
    ii. Make sure that Is Showing is unchecked.

7. Go back up to the Branch and drag the True component to the previously placed Delay, with a Duration of .5. Make sure the Delay is still connected to the Reverse of Sliding Door Timeline.
    i. EX:

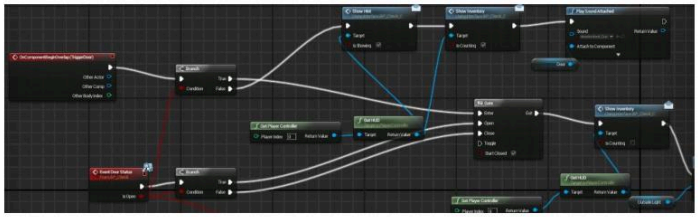


8. Go up to Event Door Status. Place a new Branch and connect the Output and Is Open to the Input and Condition of the Branch.

9. Right Click and create a Gate.
    i. Drag the True of Branch into Open and the False into Close.

10. Go up to OnComponentBeginOverlap and drag the Output to place another Branch.
    i. Drag True to the Enter of Gate.

ii. Drag False and place the Interface Messages of ShowHint.
    a. Make sure Is Showing is Checked.
iii. Drag the Output and place the Interface Messages of Show Inventory.
    a. Make sure Is Counting is Checked.
iv. Right Click and place Get Player Controller. Drag the Return Value and place Get HUD. Drag the Return Value into the Target of Show Hint and Show Inventory.
v. Drag the Output of Show Inventory and place Play Sound Attached.
    a. Set Sound to doorlocked_Cue.
    b. Right Click and place a Get of Door. Connect it to Attach to Component.
vi. EX:



11. Drag the Exit of Gate to place the Interface Messages of Show Inventory.
    i. Right Click and place Get Player Controller. Drag the Return Value and place Get HUD. Drag the Return Value into Target.
    ii. Make sure Is Counting is Unchecked.

12. Place a reference to the Outside Light from the blueprint. Drag the Output and place Set Light Color.
    i. Make the Input of Set Light Color is connected to the Output of Show Inventory.
    ii. Select the color swatch of New Light Color and choose a green value.
    iii. Connect the Output of Set Light Color into the Play of Sliding Door Timeline.

13. Go into Components and change the Outside Light color to a red value.

14. Drag the Output of Set Relative Location, after the Sliding Door Timeline, and place Play Sound Attached.
    i. Set Sound to slidingdoor_Cue.
    ii. Place a Get of Door and connect it to Attach to Component.
    iii. This will play the sound continuously until the door stops updating. If you are using a different sound that you would like to play once then attach it before the Open and Reverse chains.

15. Compile and Save the code before opening back up the Level Blueprint.

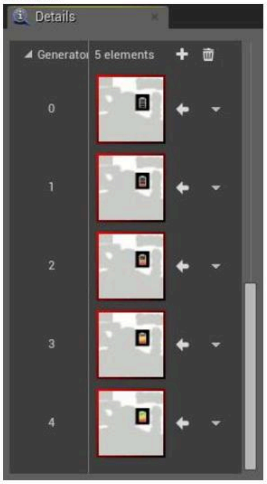
16. In the Find a Node section find and place Set Texture Parameter Value.
    i. Connect the Output of the Add Charge node to the Input.
    ii. Place a Get of M_GeneratorInstance and connect it to Target.

iii. Create a Texture2D variable. Select the box of squares next to the Variable Type to create an array. Name it GeneratorLevel.

a. Compile.

b. In Details of the GeneratorLevel, create 5 elements. Place the different textures for the generator within them.

c. EX:



iv. Place a Get of GeneratorLevel.

a. Drag the Output and place a GET.

b. Place the Get of Collect and connect its Output to the open slot of GET.

v. Set Parameter Name to Texture_Sample_2D.

vi. Connect the Output of GET into the Value of Set Texture Parameter Value.

17. Drag the Output of Set Texture Parameter Value and place a Branch.

i. Drag Condition and place Integer >= Integer.

ii. Place a Get of Collect and attach it to the top section.

iii. Set the bottom section to 4.

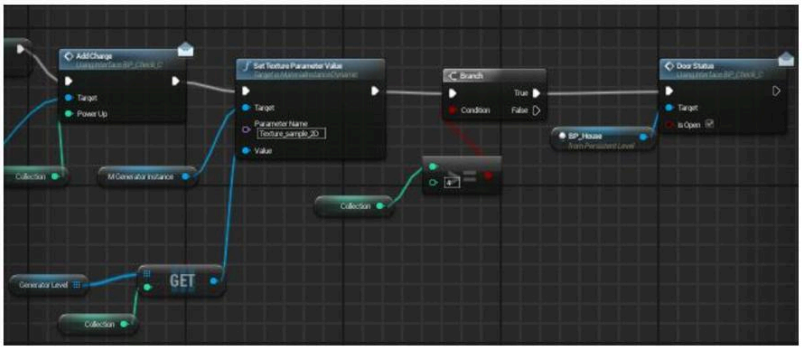
18. Drag True and place the Interface Messages of DoorStatus.

i. Select BP_House in the scene of the Editor. Go back in the Level Blueprint and Right Click to create a reference.

a. Attach the reference to Target.

ii. Make sure that Is Open is checked.

19. EX:



20. Compile and Save. When playing, the player should be able to pick up the batteries to change the generator texture and open the door when they have 4 batteries.

HUD

1. Back in the Blueprints folder, create a blueprint using the HUD parent class called FPSHUD.

2. Save and open back up your code editor for the project.

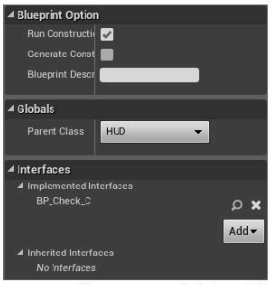
3. Open FPSGameMode.cpp.
    i. Add the following code to the constructor:

```
//May need to change reference text if FPSHUD is in another folder.
static ConstructorHelpers::FObjectFinder<UBlueprint>
HUDObject(TEXT("Blueprint'/Game/Blueprints/FPSHUD.FPSHUD'"));

    if (HUDObject.Object != NULL)
    {
        HUDClass = (UClass*)HUDObject.Object->GeneratedClass;
    }
```

4. Build and open up the unreal editor.

5. Open up FPSHUD and go to Graph.
    i. At the top of the blueprint editor, click on Blueprint Props.
        a. Under Interfaces, add BP_Check_C.



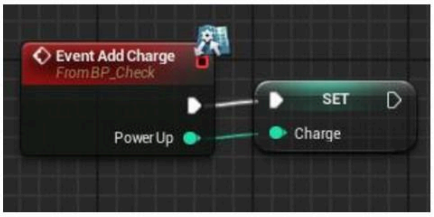
ii. Create a new Integer variable called Charge.

a. Place a Set of Charge.

iii. Right Click and place Event Add Charge.

v. Attach the Output and Power Up of Event Add Charge into the Input and Charge of Set Charge.

a. Result:

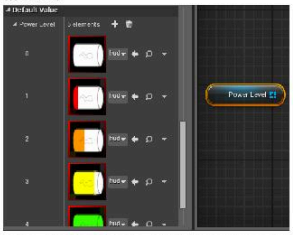


6. Create a Texture2D variable and select the checker box next to the Variable Type. Name it PowerLevel.

7. Compile and place the Get of PowerLevel on the graph.

i. Under the Default Value of PowerLevel, make 5 elements and place the battery textures for the HUD.

a. EX:



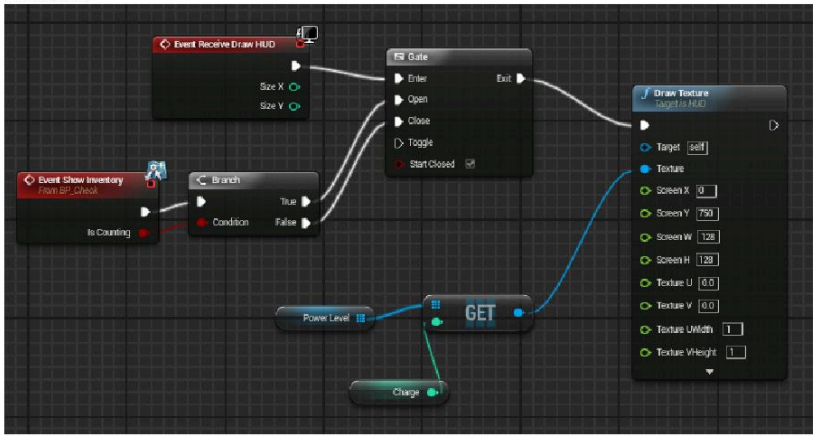
ii. Drag the Output box of PowerLevel and place a GET.

a. Place another Get of Charge and connect it to the integer input of GET.

8. Right Click and place Event Receive Draw HUD. Right Click again and place Event Show Inventory.

i. Drag the Output of Event Show Inventory and place a Branch node.
    a. Drag the Is Counting into the Condition of Branch.
ii. Drag the Output of Event Receive Draw HUD and place Gate.
iii. Drag the True of Branch into Open of Gate and False into Close.

9. Drag the Exit of Gate and place Draw Texture.
    i. Connect the output of GET into Texture of Draw Texture.
    ii. Give Screen Y a value of 750.
    iii. Give Screen W and Screen H value of 128.
    iv. Give Texture UWidth and Texture VHeight a value of 1.
    v. Result:

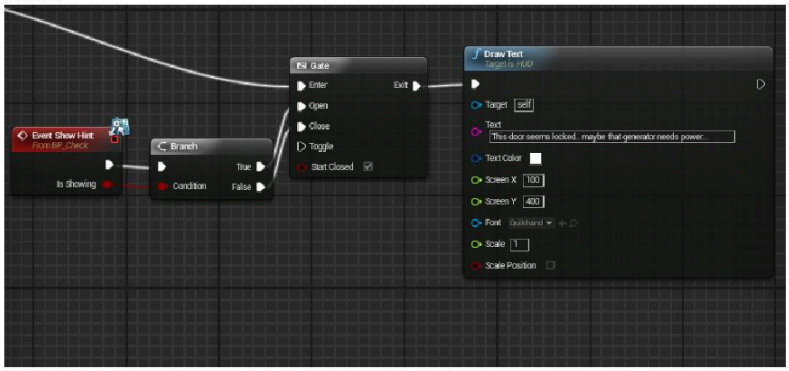


11. Drag the Output of Draw Texture and place a Gate.
    i. Right Click and place Event Show Hint.
    ii. Create a Branch and connect the Output and Is Showing of Event Show Hint into the Input and Condition.
    iii. Connect True to Open and False to Close.

12. Compile and Save. Go back to the editor and create a Fonts folder.
    i. Go New > Materials & Textures > Font.
        a. This uses fonts installed on the computer
    ii. Name it after the font used.

13. Open back up FPSHUD. Drag the Exit of Gate and place Draw Text.
    i. Using the following for Text: The door seems locked… maybe the generator needs power…
    ii. Show white for the Text Color.
    iii. Make the text appear in the center of the screen. I used Screen X 100 and Screen Y 400.
    iv. Set Font to the one created.

v. EX:



14. Compile and Save.

Result

The character should be unable to walk into the house until they have collected all four batteries. The player should be given a hint at the door and they should be able to see their progress in collecting batteries. There should be a visual cue from the HUD, generator, and outside house light that indicates progress.

Submission

Packaged projects will be checked in class on Monday (11/3/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. For more fonts search here: http://www.dafont.com/

CGT 345

Due Monday (11/3/14)

Goal

Have the player collect batteries in order to open the outpost's door.

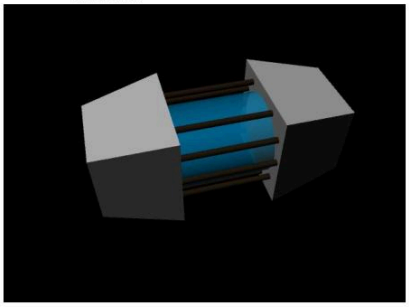
Setup

1. Create a copy of you previous project.

2. Create a Sounds folder and import the sound files.
    i. You can also find your own here: http://www.freesound.org/
    ii. Make sure you have a Door Lock, Door Open, and Collect sound.

Power Cell

1. Model a power cell.
    i. It should roughly look like a battery, fuel, or something that can power a generator.
    ii. Can UV unwrap and texture but solid colors are okay.
    iii. EX:

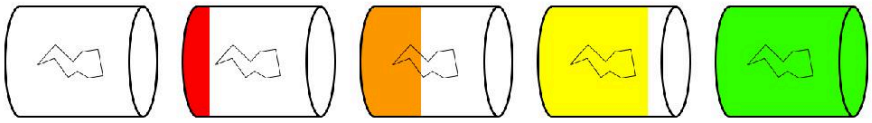


2. Import the mesh into the Models folder in the editor.
    i. Make sure that Import Material is checked under Advance.

HUD Elements

1. Create a heads up display that will show the amount of power cells collected.

i. Keep the dimensions a power of 2 (256x256 or 512x512).
ii. Make a 5 different power levels to match the generator.
iii. EX:



2. Save them as a PNG file type, or another file type that supports transparency.
i. When saving, make sure they have names that are easy to reference later. (power0, power1…power4)

3. Import the textures into the editor within the Textures folder.

Sound

1. Open your Sounds folder.
i. Right Click doorlocked and select Create Cue.
a. Double Click to open.
b. Disconnect the two nodes.
c. Drag the Output of Wave Player and place Modulator.
A. Set Pitch Min to .7 and Pitch Max to 1.25.
d. Drag the Output of Modulator into Output.
e. Save.
ii. Right Click slidingdoor and select Create Cue.
a. Double Click to open.
b. Disconnect the two nodes.
c. Drag the Output of Wave Player and place Modulator.
A. Set Pitch Min to 1.5 and Pitch Max to 2.0.
d. Drag the Output of Modulator into Output.
e. Save.

Generator

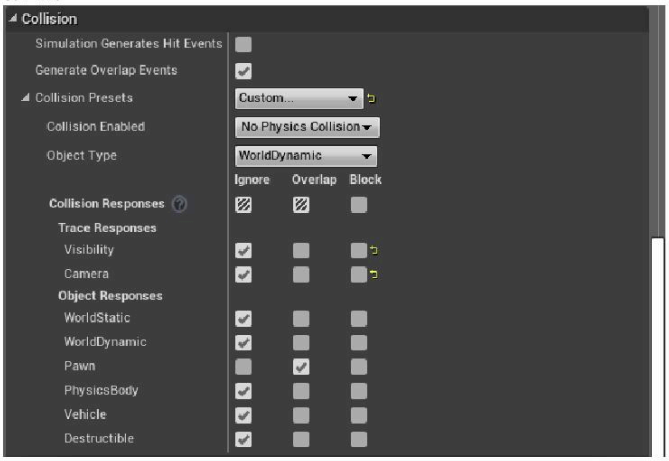
1. Open the material for the generator.
i. Right Click the texture and select Convert to Parameter.
ii. Name it Texture_Sample_2D.

2. Compile and Save.

Pick Up

1. Create a Blueprint using the Actor class called Battery.

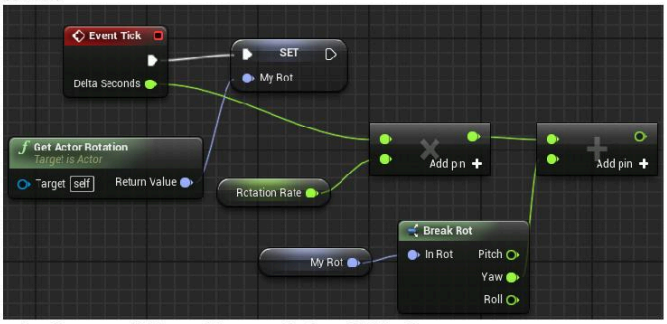
2. Open Battery and go to the Components section.
    i. Create a sphere component called BaseCollisionComponent.
        a. Under Details, change the Sphere Radius, under Shape, to 60.
        b. In Details under Collision, Change the Collision Presets to Custom and Ignore all collisions except for Pawn, which should have Overlap checked.
        c. EX:



    ii. Create a static mesh component called PickupMesh.
        a. In Details, set the static mesh to the battery model.
        b. Under Collision, Change the Collision Presets to Custom and Ignore all collisions.

3. Go to the Graph section of Battery.
    i. Create a new float variable called RotationRate.
        a. Compile and Save.
        b. Give RotationRate a Default Value of 180.
    ii. In Find a Node, search and place Get Actor Rotation.
        a. Right Click the Return Value and Promote to Variable.
        b. Name the variable MyRot.
    iii. Right Click and place the Event Tick.
        a. Connect the Output arrow of Tick to the Input arrow of Set MyRot.
    iv. Place a Get of RotationRate.
        a. Drag the Output and place Float*Float.
        b. Drag the Delta Seconds Output of Event Tick into the second slot of Float*Float.
        c. Drag the Output of Float*Float and place Float+Float.
    v. Place a Get of MyRot.
        a. Drag the Output and place Break Rot.
        b. Drag the Yaw of Break Rot into the open slot of Float+Float.

c. EX:



vi. Drag the Output of Float+Float and place Make Rot.

a. Make sure the Output of Float+Float is in the Yaw of Make Rot.

b. Drag the Return Value of Make Rot and place a Set of MyRot.

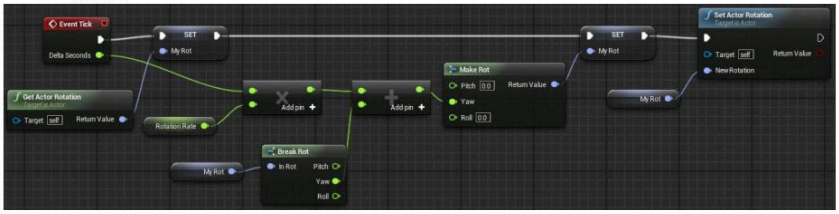
c. Go back to the first Set MyRot and connect its Output to the Input of the new Set MyRot.

d. Drag the Output of Set MyRot and place Set Actor Location.

e. Place a Get of MyRot and connect it to the New Rotation of Set Actor Location.

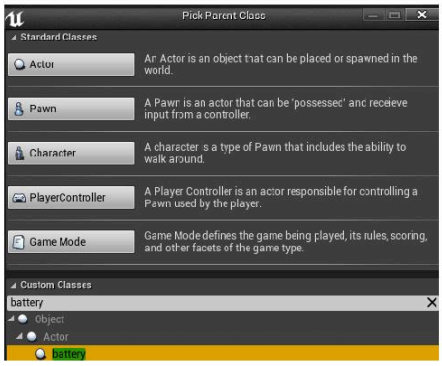
4. Compile and Save.

i. EX:



5. Go to the Blueprints folder and create a Battery Custom Class Blueprint called BP_Battery.

i. EX:

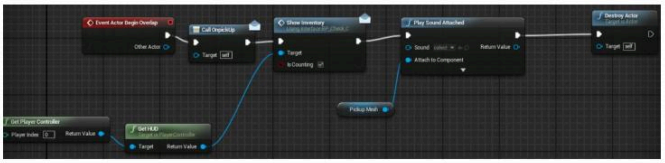


6. In the Blueprints folder, select New > Miscellaneous > Blueprint Interface. Name it BP_Check.

7. Open BP_Check.
    i. Select the + Function button. Name the new function ShowHint.
        a. In the details of ShowHint, Press New within the Inputs section and name the Boolean variable bIsShowing.
    ii. Select the + Function and name the new function ShowInventory.
        a. Add an Input variable called bIsCounting.
    iii. Select the + Function again and name the new function AddCharge.
        a. Add an Integer for Inputs called PowerUp.
    iv. Add a Function Called DoorStatus.
        a. Make a Boolean input called bIsOpen.

8. Compile and Save BP_Check.

9. Open BP_Battery and go to the Graph section.
    i. In the My Blueprints tab, click on Event Dispatcher. Name the event OnPickUp.
        a. It should be to the right of Macro and Graph.
    ii. Right Click and search overlap. Select the event Event Actor Begin Overlap.
    iii. Drag the arrow and find Call OnPickUp.
    iv. Drag the arrow and place Show Inventory Interface Message.
        a. Make sure that Is Counting is checked within Show Inventory.
        b. Right Click and place Get Player Controller.
        c. Drag the Return Value and place Get HUD.
        d. Drag the Return Value of Get HUD into the Target.
    v. Drag the Output of Show Inventory and place Play Sound Attached.
        a. Set Sound to collect.
        b. Right Click and place a Get of PickupMesh.
        c. Connect to Attach to Component.
    vi. Drag the Output of Play Sound Attached and place Destroy Actor.
        a. End Result:



10. Compile and Save BP_Battery.

Generator Instance

1. Open the Level Blueprint.

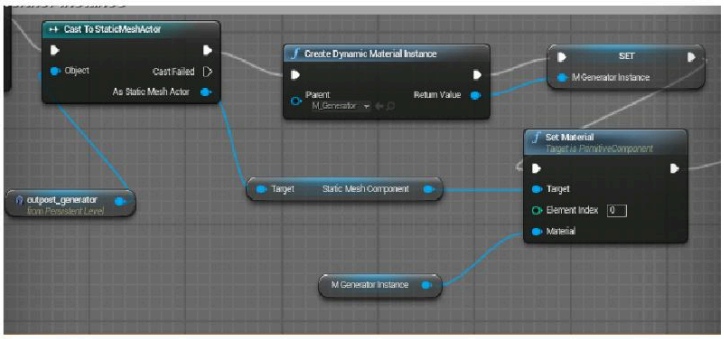
2. Create a new variable with the type actor.
   ii. Place a Get of it on the scene.
   iii. Drag and place Cast To StaticMeshActor.
      a. You can delete the actor variable as we will not be using it.
   iv. Drag the input arrow of Cast To StaticMeshActor into the output of the last Execute Console Command Node.

3. Go back to the scene and select you Generator mesh. In the Level Blueprint, Right Click and add a reference to the mesh.
   i. Connect it to the Object of Cast To StaticMeshActor.

4. The top Output of Cast To StaticMeshActor and place Create Dynamic Material Instance.
   i. For the Parent of Create Dynamic Material Instance, choose M_Generator.
   ii. For the Cast Failed output, you can connect to Print String to check if the code is working.
5. Right Click the Return Value of Create Dynamic Material Instance and select Promote to Variable. Name the new variable M_GeneratorInstance.

6. Search in Find a Node and place a Set Material Node.
   i. Connect the Set of M_GeneratorInstance into the top input of Set Material.
   ii. Drag the As Static Mesh Actor component of Cast To StaticMeshActor into the Target of Set Material.
   iii. Place a Get of the M_GeneratorInstance variable and connect it to Material.

7. This will create an instance of the Generator material that can have its texture changed during runtime.



Spawning

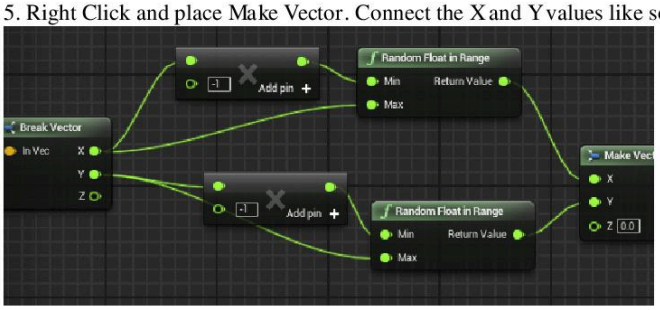
1. Create a trigger volume in the scene and name it BatterySpawn.
    a. The box trigger can be found in the Modes tab under Place > Basic.

2. Scale BatterySpawn in the X and Y direction so that it covers a large area.
    a. Make sure the House or an unreachable area are not within the volume.

3. Make sure that BatterySpawn is selected and go back to the Level Blueprint. Right Click and place a reference of the object.
    i. Drag the Output and place Get Actor Bounds.
    ii. Drag Box Extent and place Break Vector.

4. Right Click and place Random Float in Range.
    i. Place Float * Float and connect it to the Min.
        a. Make one of the values is -1.
    ii. Connect the X value of Break Vector into the empty spot, the spot that isn't -1 of Float * Float, and into the Max of Random Float in Range.
    iii. Do the same for Y.

5. Right Click and place Make Vector. Connect the X and Y values like so:



6. Create another reference to BatterySpawn and drag its Output to place Get Actor Location.
    i. Drag Return Value of Get Actor Location and place Vector + Vector.
    ii. Drag the Return Value of Make Vector into the open spot of Vector + Vector.

7. Place Make Transform. Connect the Output of Vector + Vector into Location.

8. Go back up to Set Material and drag its Output to place a ForLoop.
    i. Set the First Index to 0 and Last Index to 3.

9. Drag Loop Body and place Spawn Actor from Class.
    i. Set Class to BP_Battery.
    ii. Connect the Return Value of the Make Transform node into Spawn Transform.

10. Create a new variable using the Variable Type BP_Battery_C.
    i. Drag a Get of it onto the graph.
    ii. Drag the Output and place Bind Event to OnPickUp.
    iii. Delete the variable.

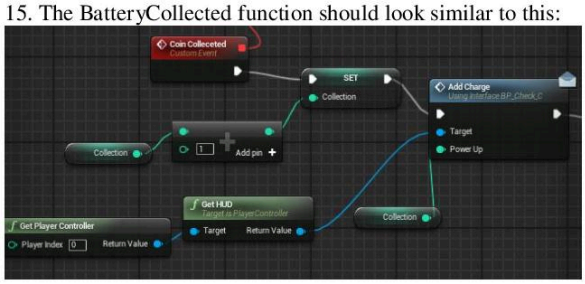
11. Connect the Output and Return Value of SpawnActor BP_Battery_C, formerly Spawn Actor from Class, into the Input and Target of Bind Event to OnPickUp.

12. Create a new Custom Event and name it BatteryCollected.
    i. Place BatteryCollected and connect the red box to Event of Bind Event to OnPickUp.

13. Create a new Integer called Collect. Place a Get and Set of the variable onto the graph.
    i. Drag the Output of Get and place Integer + Integer. Make the value of the open slot 1.
    ii. Connect the Output to the Collect of Set.
    iii. Connect the Output of BatteryCollected into the Input of Set Collect.

14. Drag the Output of Set Collection and place the Interface Messages AddCharge.
    i. Right Click and place Get Player Controller. Drag the Return Value and place Get HUD. Drag its Return Value to Target.
    ii. Place a Get of Collect and connect it to Power Up.

15. The BatteryCollected function should look similar to this:



16. Compile the Level Blueprint. On Play, you should see 4 batteries spawn and then disappear when the character walks through them.

Close and Open

1. Open the BP_House blueprint.

2. Go to Graph and select Blueprint Props. Under Interfaces, Add BP_Check_C.

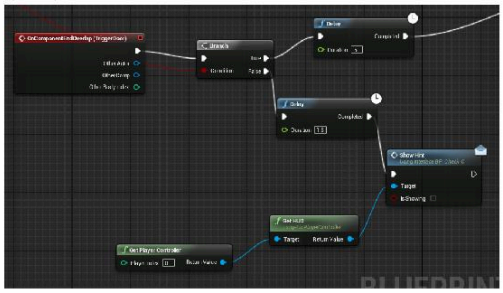
3. Disconnect Begin Overlap and End Overlap event nodes. Move them to the left and Right Click between them and place Event Door Status.
    i. Make sure Event Door Status is in a good spot. We will be using the Is Open component frequently for this graph.

4. Go down to OnComponentEndOverlap and drag the top Output to place a Branch.
    i. Drag the Is Open of Event Door Status into the Condition of the Branch.

5. Drag False and place a Delay node with a Duration of 1.5.

6. Drag the Completed of Delay to place the Interface Messages of ShowHint.
    i. Right Click and place Get Player Controller. Drag Return Value and place Get HUD. Connect the Return Value to Target.
    ii. Make sure that Is Showing is unchecked.

7. Go back up to the Branch and drag the True component to the previously placed Delay, with a Duration of .5. Make sure the Delay is still connected to the Reverse of Sliding Door Timeline.
    i. EX:

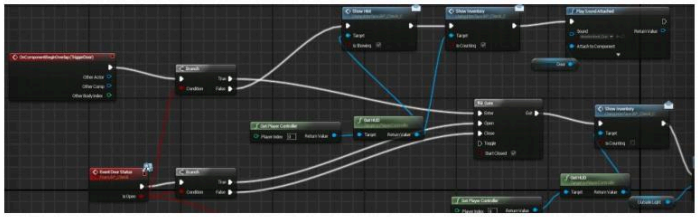


8. Go up to Event Door Status. Place a new Branch and connect the Output and Is Open to the Input and Condition of the Branch.

9. Right Click and create a Gate.
    i. Drag the True of Branch into Open and the False into Close.

10. Go up to OnComponentBeginOverlap and drag the Output to place another Branch.
    i. Drag True to the Enter of Gate.

ii. Drag False and place the Interface Messages of ShowHint.
    a. Make sure Is Showing is Checked.
iii. Drag the Output and place the Interface Messages of Show Inventory.
    a. Make sure Is Counting is Checked.
iv. Right Click and place Get Player Controller. Drag the Return Value and place Get HUD. Drag the Return Value into the Target of Show Hint and Show Inventory.
v. Drag the Output of Show Inventory and place Play Sound Attached.
    a. Set Sound to doorlocked_Cue.
    b. Right Click and place a Get of Door. Connect it to Attach to Component.
vi. EX:



11. Drag the Exit of Gate to place the Interface Messages of Show Inventory.
    i. Right Click and place Get Player Controller. Drag the Return Value and place Get HUD. Drag the Return Value into Target.
    ii. Make sure Is Counting is Unchecked.

12. Place a reference to the Outside Light from the blueprint. Drag the Output and place Set Light Color.
    i. Make the Input of Set Light Color is connected to the Output of Show Inventory.
    ii. Select the color swatch of New Light Color and choose a green value.
    iii. Connect the Output of Set Light Color into the Play of Sliding Door Timeline.

13. Go into Components and change the Outside Light color to a red value.

14. Drag the Output of Set Relative Location, after the Sliding Door Timeline, and place Play Sound Attached.
    i. Set Sound to slidingdoor_Cue.
    ii. Place a Get of Door and connect it to Attach to Component.
    iii. This will play the sound continuously until the door stops updating. If you are using a different sound that you would like to play once then attach it before the Open and Reverse chains.

15. Compile and Save the code before opening back up the Level Blueprint.

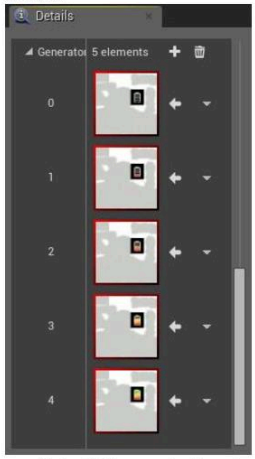
16. In the Find a Node section find and place Set Texture Parameter Value.
    i. Connect the Output of the Add Charge node to the Input.
    ii. Place a Get of M_GeneratorInstance and connect it to Target.

iii. Create a Texture2D variable. Select the box of squares next to the Variable Type to create an array. Name it GeneratorLevel.

a. Compile.

b. In Details of the GeneratorLevel, create 5 elements. Place the different textures for the generator within them.

c. EX:



iv. Place a Get of GeneratorLevel.

a. Drag the Output and place a GET.

b. Place the Get of Collect and connect its Output to the open slot of GET.

v. Set Parameter Name to Texture_Sample_2D.

vi. Connect the Output of GET into the Value of Set Texture Parameter Value.

17. Drag the Output of Set Texture Parameter Value and place a Branch.

i. Drag Condition and place Integer >= Integer.

ii. Place a Get of Collect and attach it to the top section.

iii. Set the bottom section to 4.

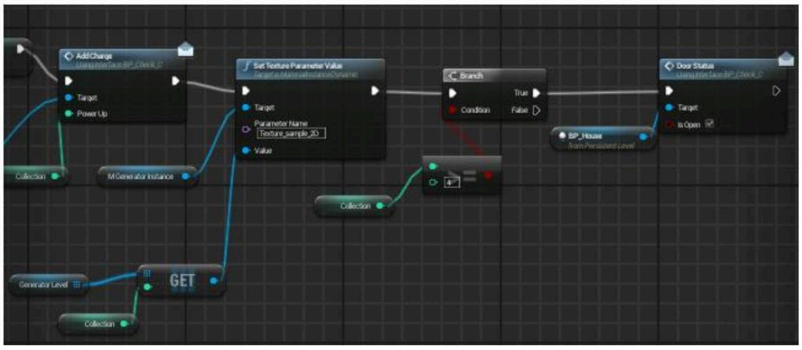
18. Drag True and place the Interface Messages of DoorStatus.

i. Select BP_House in the scene of the Editor. Go back in the Level Blueprint and Right Click to create a reference.

a. Attach the reference to Target.

ii. Make sure that Is Open is checked.

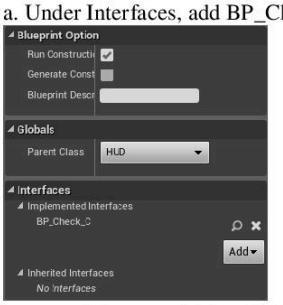
19. EX:



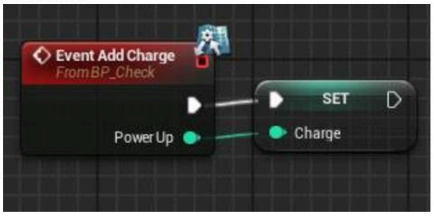
20. Compile and Save. When playing, the player should be able to pick up the batteries to change the generator texture and open the door when they have 4 batteries.

HUD

1. Open up BP_FPSHUD and go to Graph.
    i. At the top of the blueprint editor, click on Blueprint Props.
        a. Under Interfaces, add BP_Check_C.

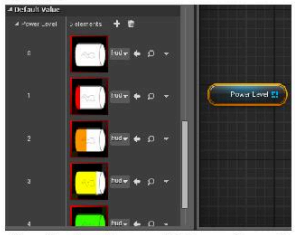


    ii. Create a new Integer variable called Charge.
        a. Place a Set of Charge.
    iii. Right Click and place Event Add Charge.
    v. Attach the Output and Power Up of Event Add Charge into the Input and Charge of Set Charge.
        a. Result:



2. Create a Texture2D variable and select the checker box next to the Variable Type. Name it PowerLevel.

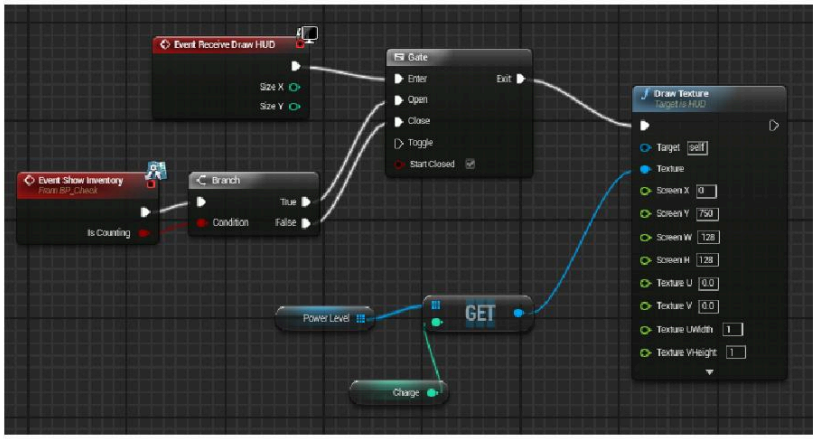
3. Compile and place the Get of PowerLevel on the graph.
    i. Under the Default Value of PowerLevel, make 5 elements and place the battery textures for the HUD.
        a. EX:



    ii. Drag the Output box of PowerLevel and place a GET.
        a. Place another Get of Charge and connect it to the integer input of GET.

4. Right Click and place Event Receive Draw HUD. Right Click again and place Event Show Inventory.
    i. Drag the Output of Event Show Inventory and place a Branch node.
        a. Drag the Is Counting into the Condition of Branch.
    ii. Drag the Output of Event Receive Draw HUD and place Gate.
    iii. Drag the True of Branch into Open of Gate and False into Close.

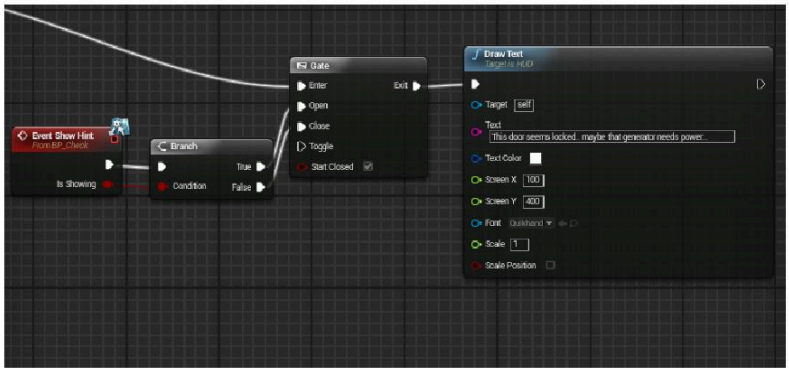
5. Drag the Exit of Gate and place Draw Texture.
    i. Connect the output of GET into Texture of Draw Texture.
    ii. Give Screen Y a value of 750.
    iii. Give Screen W and Screen H value of 128.
    iv. Give Texture UWidth and Texture VHeight a value of 1.
    v. Result:



6. Drag the Output of Draw Texture and place a Gate.
    i. Right Click and place Event Show Hint.
    ii. Create a Branch and connect the Output and Is Showing of Event Show Hint into the Input and Condition.
    iii. Connect True to Open and False to Close.

7. Compile and Save. Go back to the editor and create a Fonts folder.
    i. Go New > Materials & Textures > Font.
        a. This uses fonts installed on the computer
    ii. Name it after the font used.

8. Open back up BP_FPSHUD. Drag the Exit of Gate and place Draw Text.
    i. Using the following for Text: The door seems locked… maybe the generator needs power…
    ii. Show white for the Text Color.
    iii. Make the text appear in the center of the screen. I used Screen X 100 and Screen Y 400.
    iv. Set Font to the one created.
    v. EX:



9. Compile and Save.

Result

The character should be unable to walk into the house until they have collected all four batteries. The player should be given a hint at the door and they should be able to see their progress in collecting batteries. There should be a visual cue from the HUD, generator, and outside house light that indicates progress.

Submission

Packaged projects will be checked in class on Monday (11/3/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. For more fonts search here: http://www.dafont.com/

CGT 345

Due Monday (11/10/14)

Goal

Collect the match and light a fire.

Setup

1. Create a copy of you previous project.

2. Download the MigrateTutorial.zip file.

Matches

1. Create an image that will be used when the matches are collected.
    i. Must be appropriate in that it helps the player see that they have collected the matches.
    ii. A quick way is to do a render of the matchbox in Maya and touch up in photoshop.
    iii. Should be a size of 128x128 with a transparent background.
    iv. EX:

2. Import the campfire fbx along with the Plywood and RockSmooth textures. Also, import the fire sound.
    i. Under Advanced, make sure Combine Meshes is checked before importing.
    ii. Open the M_Rock material and make sure RockSmooth is attached to Base Color.
    iii. Open the M_Stick material and make sure Plywood is attached to Base Color.
    iv. Right Click campfire and select *Create Blueprint Using…*
        a. Create a blueprint called BP_Campfire in the Blueprints folder.
    v. Open the fire sound file.
        a. Under Compression, make sure Looping is checked.
        b. Save.

3. Open BP_Campfire.
    i. Add a sphere component called CampfireTrigger.
        a. Set the Sphere Radius to 200.

b. Set the Collision Presets to Custom with every collision response set to Ignore except for Pawn, which should be set to Overlap.

ii. Add a sphere component called CampfireCollide.
a. Set the Sphere Radius to 110.
b. Set the Collision Presets to BlockAll.

iii. Add an audio component called PlayFire.
a. Under Sound, set Sound to fire.
b. Under Attenuation, make sure Override Attenuation is checked.
c. Under Activation, make sure that Auto Activate is unchecked.

4. Import the T_SmokeTile_N, T_FireSpot, and T_Fire_Tile_03 into the Textures folder.

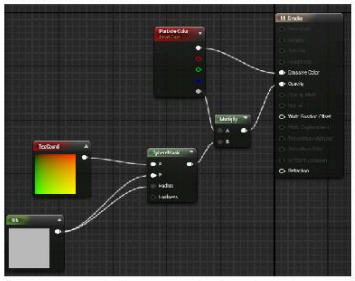
Smoke Material

1. Create new material in the Materials folder called M_Smoke.
i. In the Details panel under Material, Change the Blend Mode to Translucent.
ii. Change Lighting Model to Unlit.

2. Right Click and place Particle Color.
i. Connect the top Output to Emissive Color.

3. Right Click and place a Texture Coordinate.
i. Drag the Output and place a Sphere Mask.
ii. Hold 1 and Left Click to place a Schalar. Give it a value of 0.5.
iii. Connect the Schalar to B and Radius.

4. Drag the Output of SphereMask and place a Multiply node.
i. Drag the bottom Output of Particle Color into the open slot of Multiply.
ii. Connect the Multiply output to Opacity.
iii. EX:



Fire Material

1. Create a new material called M_Fire in the Materials folder.
i. In the Details panel under Material, Change the Blend Mode to Translucent.
ii. Change Lighting Model to Unlit.

2. Place the T_FireSpot texture and connect its red output to Opacity.

3. Right Click and place a Lerp.
    i. Place the T_Fire_Tile_03 texture and connect its red output to Alpha.
    ii. Create a VectorParameter called Cool Color. Give it a red value.
        a. Connect its top Output to A.
    iii. Create a VectorParamter called Hot Color. Give it an orange value.
        a. Connect its Output to B.

4. Drag the Output of Lerp and place Power.
    i. Select Power and change Const Exponent in the Details panel to 1.2.
    ii. Connect the Output to a Multiply.
        a. Input a value of 16 for B.
    iii. Drag the Output and place a Multiple.
    iv. Right Click and place Particle Color. Connect it to the open slot of Multiply.
    v. Connect the Output of Multiply into Emissive Color.

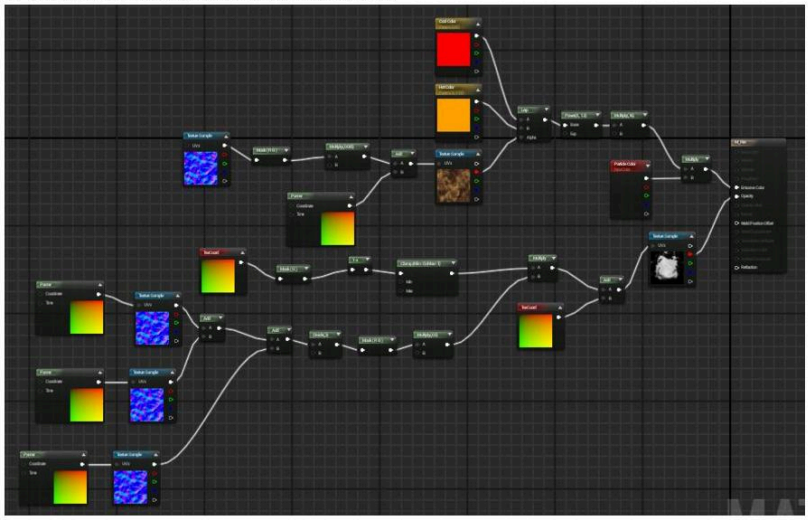
5. Go back around the T_Fire_Tile_03 texture and place T_SmokeTile_N.
    i. Drag the top Output of T_SmokeTile_N and place a ComponentMask.
    ii. Drag the Output of the Mask and place Multiply.
        a. Set the open slot to 0.025.
    iii. Right Click and place a Panner.
        a. Set Speed Y to 0.3.
    iv. Drag the Output of Multiply and place Add. Connect the Output of Panner into the other slot of Add.
    v. Connect the Output of Add into the UVs of T_Fire_Tile_03.

6. Place a Texture Coordinate.
    i. Drag the Output and place a ComponentMask.
        a. Make sure only that the Mask only takes the G value.
    ii. Drag the Output of Mask(G) and place 1-x.
    iii. Drag its Output and place a Clamp.
    iv. Drag the Clamp Output and place Multiply.
    v. Drag the Output and place Add.
        a. Right Click and place a Texture Coordinate. Connect it to the open slot of Add.
    vi. Drag the Output of Add into the UVs of T_FireSpot.

7. Go back past the open Multiply and place a Panner with a Speed Y of 0.4.
    i. Place a T_SmokeTile_N and connect the Output of Panner to its Input.
    ii. Copy the Panner and Texture Sample twice, so that there are three instances.
        a. Set the second Panner to a Speed X of .255 and a Speed Y of .5.
        b. Set the third Panner to a Speed X of -0.1258 and a Speed Y of .35.
    iii. Place an Add node and connect the Output of two Texture Samples.
        a. Place another Add and connect the Output of the previous Add and the third Texture Sample.

iv. Drag the Output of Add and place Divide.
    a. Set the open slot to 3.
v. Drag the Divide Output and place a ComponentMask.
vi. Drag the Output and place a Multiply.
    a. Set the open value to 0.5.
vii. Connect the Output to the open slot of Multiply.

8. Save. The material should similar to this:



Campfire Flame

1. Create a new folder called Particles.

2. In the new folder, create a particle system called Campfire. Double click to open Cascade.

3. On top of the Emitter, Right Click > Emitter > Rename Emitter. Name it Smoke.
    i. Select Required.
        a. Change the Material of Emitter to M_Smoke.
        b. Set the Z value of Emitter Origin to 40.0.
    ii. Select Spawn.
        a. Go Rate > Distribution and set the Constant to 8.0.
    iii. Select Lifetime.
        a. Set the Min and Max value to 6.0.
    iv. Select Initial Size.
        a. Set the Max X, Y, and Z value to 150.0.
        b. Set the Min X, Y, and Z value to 100.0.
    v. Select Initial Velocity.

a. Under Start Velocity, set the Distribution to DistributionVectorConstant.
b. Make sure X is 10, Y is 10, and Z is 100.

vi. Select Color Over Life.
a. Go to Color Over Life > Distribution > Constant Curve > Points. Make sure there are 4 elements for Points.
A. Set 0 to the In Val of 0.0 and the Out Val of 0.0, 0.0, 0.0.
B. Set 1 to the In Val of .35 and the Out Val of 0.1, 0.1, 0.1.
C. Set 2 to the In Val of .85 and the Out Val of 0.8, 0.8, 0.8.
D. Set 3 to the In Val of 1.0 and the Out Val of 1.0, 1.0, 1.0.
b. Go to Alpha Over Life > Distribution > Constant Curve > Points. Make sure there are 2 elements for Points.
A. Set 0 to an In Val of 0.0 and Out Val of 1.0.
B. Set 1 to an In Val of .75 and Out Val of .75.

vii. Right Click Smoke and go Size > Size By Life. Select Size By Life.
a. Go Life Multiplier > Distribution > Constant Curve > Points. Make sure Points has 2 elements.
A. Set 0 to the In Val of 0.0 and Out Val of 0.5, 0.5, 0.5.
B. Set 1 to the In Val of 2.0 and Out Val of 5.0, 5.0, 5.0.

4. Right Click to the right of Smoke and select New Particle Sprite Emitter. Name it Fire.
i. Select Required.
a. Select M_Fire for the Material.
ii. Select Spawn.
a. Go to Rate > Distribution and set the Constant to 5.0.
iii. Select Lifetime.
a. Set Max to 1.25.
iv. Select Initial Size.
a. Set Max X, Y, and Z to 220.
b. Set Min X, Y, and Z to 150.
v. Select Initial Velocity.
a. Set Max Z to 35.
b. Set Min Z to 25.
vi. Select Color Over Life.
a. For Color Over Life, make sure both of the Points have an Out Val of 1.0.
b. For Alpha Over Life, make sure the Out Val for 0 is 1.0 and the Out Val for 1 is 0.0.
vii. Optional: Right Click Fire and go Light >Light.

5. Save.

Campfire

1. Open BP_Check.
i. Add a Function called MatchHud.
a. Add a bool Input called bIsHolding.
ii. Add a Function called ShowFireHint.

a. Add a bool Input called bIsWithout.

iii. Compile and Save.

2. Open BP_CampFire and go to Graph.

i. Select Blueprint Props.

a. Add BP_Check_C in Interfaces.

3. Add a bool called bIsChecking.

i. Right Click and place Event MatchHud.

ii. Place a Set of bIsChecking.

iii. Connect the Output and Is Holding to Set bIsChecking.

4. Select CampfireTrigger.

i. Place an OnComponentBeginOverlap with CampfireTrigger.

a. Drag the Output arrow and place a Branch.

A. Connect a Get of bIsChecking into Condition.

b. Drag the False of Branch and place the Interface Message of Show Fire Hint.

A. Right Click and place Get Player Controller.

B. Drag the Return Value and place Get HUD.

C. Drag the Return Value into Target.

D. Make sure bIsWithout is checked.

c. Place a Get of PlayFire.

A. Drag the Output and place Play.

B. Set Start Time to 1.

C. Connect True to the Input.

d. Drag Output for Play and place Spawn Emitter at Location.

A. Select Campfire for Emitter Template.

B. Right Click and place Get Actor Location. Drag the Return Value into Location.

e. Drag the Spawn Emitter Output arrow and place the Interface Message of Match Hud.

A. Right Click and place Get Player Controller.

B. Drag the Return Value and place Get HUD.

C. Drag the Return Value into Target.

D. Make sure Is Holding is Unchecked.

ii. Place an OnComponentEndOverlap with CampfireTrigger.

a. Drag the Output arrow and place a Branch.

A. Connect a Get of bIsChecking into Condition.

b. Drag False and place the Interface Message of Show Fire Hint.

A. Right Click and place Get Player Controller.

B. Drag the Return Value and place Get HUD.

C. Connect the Return Value into Target.

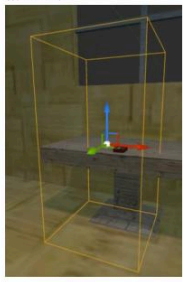
D. Make sure bIsWithout is Unchecked.

5. Compile and Save.

6. Place BP_Campfire in the scene.

Matchbox Pick Up

1. Open BP_House and go to Components.
    i. Add a Box Component and name it MatchPickUp.
    ii. Set the Collision Preset to Custom.
        a. Set Everything to Ignore expect for Pawn that should set to Overlap.
    iii. Set up MatchPickUp so that it covers the area in front of the matchbox.
        a. EX:



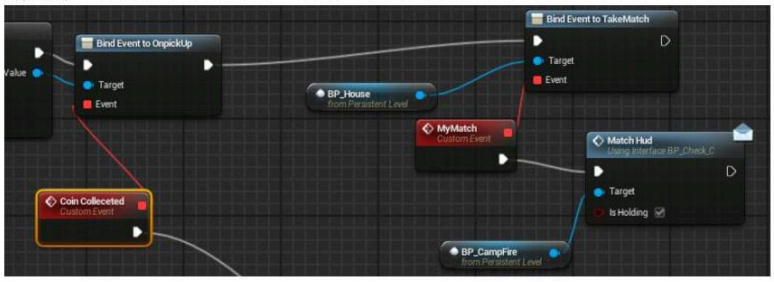
2. Go to Graph with MatchPickUp selected.
    i. Create an Event Dispatcher called TakeMatch.
    ii. Place OnComponentBeginOverlap using MatchPickUp.
    iii. Drag the Output and place Call TakeMatch.
    iv. Drag the Output and place Destroy Component.
        a. Connect the Target to your Matchbox component.
    v. Drag the Output and place the Interface Message of MatchHud.
        a. Right Click and place Get Player Controller.
        b. Drag the Return Value and place Get HUD.
        c. Drag the Return value into Target.
        d. Make sure that Is Holding is checked.
    vi. Drag the Output and place Play Sound Attached.
        a. Set Sound to collect.
        b. Place a Get of Table and connect to Attach to Component.

3. Compile and Save.

4. Select BP_House in the scene and open the Level Blueprint.
    i. Place a reference of BP_House and Bind Event to TakeMatch.
        a. Name the Custom Event MyMatch.
    ii. Connect the Input to the Bind Event to the Output of the previous Bind Event.
    iii. Drag the Output of MyMatch and place the Interface Message of MatchHud.
        a. Select BP_Campfire in the scene and create a reference of it in the Level Blueprint.
        b. Connect BP_Campfire to Target.

c. Make sure that Is Holding is checked.

iv. EX:



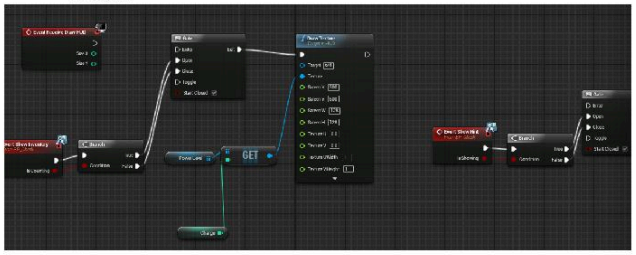
5. Compile and Save.

HUD

1. Open FPSHUD and go to Graph.

i. Disconnect the Enter of both Gates.

a. EX:

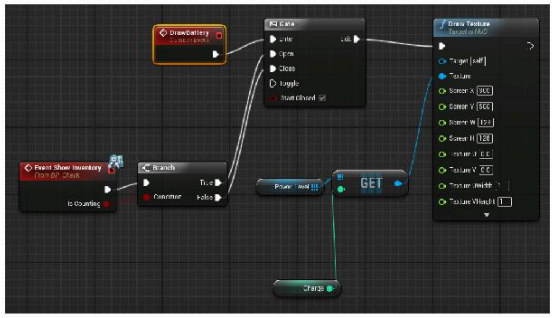


ii. Drag Event Receive Draw HUD away for the moment.

iii. Right Click and create a custom event. Name the event DrawBattery.

a. Drag the Output arrow into the Enter of the Gate for the Draw Texture chain.

b. EX:



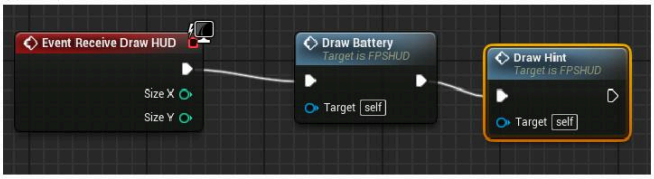
iv. Right Click and create the custom event. Name the event DrawHint.

a. Drag the Output arrow into the Enter of the Gate for the Draw Text chain.

v. Drag the Output of Event Draw Receive HUD and place the Call Function called DrawBattery.

a. Drag the Output of DrawBattery and place the Call Function called DrawHint.

b. EX:



vi. Place a Custom Event called WinHint.

vii. Drag the Output and place a Gate.

a. Right Click and place the Event ShowFireHint.

b. Drag the Output and place a Branch.

c. Connect bIsWithout to Condition.

d. Connect True to Open and False to Close.

viii. Drag the Output and place Draw Text.

a. Set the Text to: *Maybe I can use this to signal help…*

b. Set Text Color to white.

c. Set the Font.

2. Create a Custom Event called MatchCollect.

i. Drag the Output and place a Gate.

a. Right Click and place Event MatchHud.

b. Drag the Output and place a Branch.

c. Connect Is Holding to Condition.

d. Connect True to Open and False to Close.

ii. Drag the Output and place Draw Texture.

a. Set the Texture to your Match Collection Texture.

b. Set Screen W and H to a value similar to battery collect.

c. Make sure Texture UWidth and Texture UHeight are set to 1.

3. Go to Event Receive Draw HUD.

i. Drag the end Output and place WinHint.

ii. Drag the Output of WinHint and place MatchCollect.

4. Compile and Save.

Result

The player will now receive a hint when they approach the campfire. Once inside the house, the player will be able to collect the match and light the campfire outside.

Submission

Packaged projects will be checked in class on Monday (11/10/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Cascade: https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/Cascade/index.html

CGT 345

Due Monday (11/17/14)

Goal

Shoot down the targets and collect the prize.

Setup

1. Create a copy of you previous project.

2. Import the two audio files and find two files for Shack Music and Win Music.
    i. Again if you would like to find your own music search here: http://www.freesound.org/

3. Import the crosshair.tga.

Sound

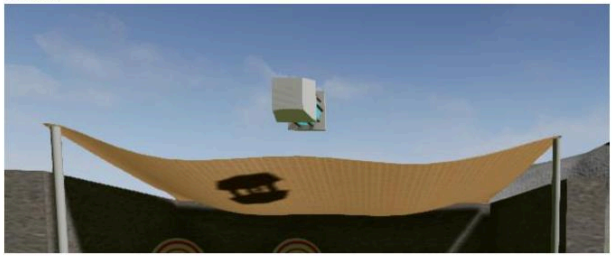
1. Import the Audio files into Sounds.
    i. Double Click the Shack Music.
        a. Under Compression make sure Looping is checked.
    i. Right Click on Swoosh and select Create Cue.
        a. Drag the Output of Wave Player and place a Modulator.
        b. Set Pitch Min to 0.6 and Pitch Max to 1.5.
        c. Drag the Output of Modulator into Output.
    ii. Right Click on thump and select Create Cue.
        a. Drag the Output of Wave Player and place a Modulator.
        b. Set Pitch Min to 0.7 and Pitch Max to 1.5.
        c. Drag the Output of Modulator into Output.

2. Drag the Shack Music onto the scene.
    i. Make sure Allow Spatialization is unchecked under Attenuation.
    ii. Make sure Auto Activate is unchecked under Activation.
    iii. Make sure Volume Multiplier is .2.

Projectile

1. Follow the tutorial from Adding Projectiles and Shooting to Projectiles Interacting with the World, stop before Adding Crosshairs:
https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial#Adding_Projectiles_and_Shooting
    i. When it asks for you to use the Sphere Mesh, use the Coconut Mesh that you created in the previous exercise.
        a. Use settings that will allow the mesh to fit in the sphere with the proper alignment. Remember in Unreal that X is forward.
    ii. When the tutorial references adding #define to FPSProject.h, add it to Island.h instead.
    iii. You do not need to follow the steps involving SM_Template_Map_Floor.

2. In the scene, place the BP_Battery blueprint above the shooting range.
    i. EX:



Shooting Zone

1. Go into FPSCharacter.h and add the following:

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Play)
bool GameOn;
```

2. Go to FPSCharacter.cpp.
    i. Add the following to the constructor:

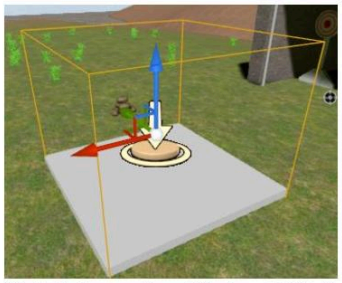
```
GameOn = false;
```

    ii. In void AFPSCharacter::OnFire() have the if statement look like the following:

```
if (ProjectileClass != NULL && GameOn == true)
```

3. Compile and open the editor.

4. Create a box trigger and have it cover the mat mesh. Name it ShackStart.
    i. You can drag ShackStart onto the mat mesh in order to parent it in the Scene Outliner.
    ii. EX:



iii. In between ShackStart and BP_Shack, place a target point called WinBattery.
    a. Make Sure it is above the terrain as we will be using this to spawn the battery.

5. Open BP_Check.
    i. Create a new function called GameTime.
        a. Add a bool to Inputs called bIsPlaying.
    ii. Create a new function called ShowTargetHint.
        a. Add a bool to Inputs called bIsTrying.
    iii. Compile and Save.
        a. Make sure to also save in the Content Browser as blueprints connected to BP_Check will change.

6. Make sure that ShackStart is selected in the scene and open the Level Blueprint.
    i. Right Click and add OnActorBeginOverlap for ShackStart.
        a. Drag and place the Interface Message called GameTime.
        b. Right Click and place Get Player Pawn. Connect the Return Value to Target.
        c. Make sure Is Playing is checked.
    ii. Drag and place another GameTime Interface Message.
        a. Right Click and place Get Player Controller.
        b. Drag the Return Value and place Get HUD.
        c. Drag the Return Value into Target.
        d. Make Sure the Is Playing is checked.
    iii. Drag and place the Interface Message of Show Target Hint.
        a. Right Click and place Get Player Controller.
        b. Drag the Return Value and place Get HUD.
        c. Drag Return Value into Target.
        d. Create a bool called bHasWon. Place a Get of the variable and connect it to Is Trying.
    iv. Select the Shack Music in the scene and place a Reference in the Level Blueprint.
        i. Drag the Output of Show Target Hint and place a Branch.
            a. Place a Get of bHasWon and connect it to Condition.
        ii. Drag the Output of Shack Music and place Play.
        iii. Connect False to the Input of Play.
    v. Right Click and place an OnActorEndOverlap for ShackStart.
        a. Place another Reference to Shack Music and drag its Output to place Stop.
        a. Place another two Interface Messages for GameTime and create a chain.

b. Make sure one Targets the Pawn while the other Targets the HUD.
c. Make is Is Playing is unchecked for both.

7. Compile and open BP_FPSCharacter.
i. Go to Graph and select Blueprint Props.
a. Under Interfaces add BP_Check_C.
ii. Right Click and place Event Game Time.
a. Drag the output and place Set Game On, from our code.
b. Connect Is Playing to the Game On of Set.
iii. Right Click and place Event Left Mouse Button.
a. Drag the Output and place a Branch.
A. Place a Get of GameOn and connect it to Condition.
b. Drag True and place Play Sound Attached.
A. Set Sound to swoosh_Cue.
B. Place a Get of Mesh and connect it Attach to Component.

8. Compile and Save. On play, the character will only be able to fire projectiles on the mat.

HUD

1. Open the FPSHUD blueprint. We will now make some changes to straighten the graph.

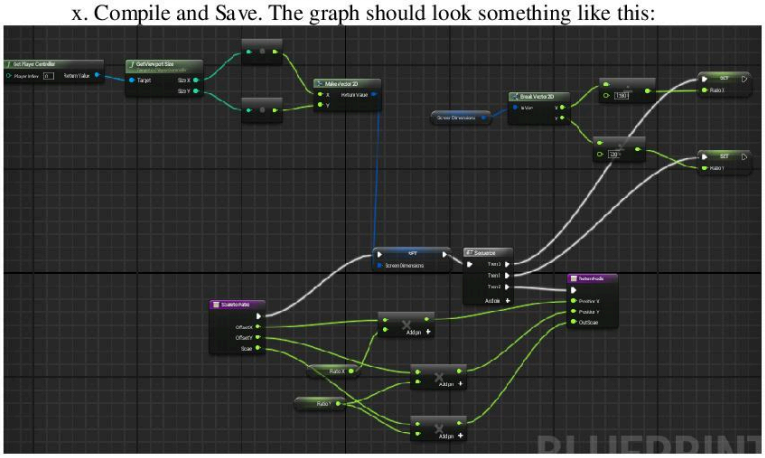
2. First create a new Function and call it ScaleToRatio. Double Click it in My Blueprint to open its graph.
i. Create 3 floats under Inputs called OffsetX, OffsetY, and Scale.
ii. Create 3 floats under Outputs called PositionX, PositionY, and OutScale.
iii. Go to a section top left of the Scale to Ratio node. Right Click and place Get Player Controller.
a. Drag the Return Value and place Get Viewport Size.
iv. Right Click and place Make Vector 2D.
a. Connect Size X and Size Y to the X and Y of Make Vector 2D.
v. Right Click the Return Value of Make Vector 2D and Promote to Variable. Name it ScreenDimensions.
vi. Drag Set ScreenDimensions down and drag the Output arrow of Scale to Ratio into the Input arrow.
vii. Drag the Output of Set ScreenDimensions and place a Sequence. Make sure the Output goes to Then 2.
a. Above Sequence place a Get of ScreenDimensions.
b. Drag the Output and place Break Vector 2D.
c. Use a Float/Float node for X and Y. Divide X by 1280 and Y by 720.
d. Create 2 float variables called RatioX and RatioY.
e. Create a Set of each and have the X divide Output connect to RatioX while the other to RatioY.
viii. Connect the Then 0 of Sequence to the Input of Set RatioX, Then 1 to the Input of Set RatioY, and Then 2 to the Input arrow of ReturnNode.

ix. Connect OffsetX, OffsetY, and Scale of Scale to Ratio to each of their own Float*Float nodes.

a. Place a Get of RatioX and connect it to the Float*Float of OffsetX. Connect the Output to PositionX of ReturnNode.

b. Place a Get of RatioY and connect it to the Float*Float of OffsetY and Scale. Drag the Output from OffsetY multiplier into PositionY and the Output from the Scale multiplier into OutScale.

x. Compile and Save. The graph should look something like this:



3. Go back to Event Graph.

i. Disconnect from the Exit of Gate from each. Place the function ScaleToRatio between the Gate and Draw node.

a. Connect the Exit of Gate into the Input of Scale to Ratio. Connect the Output to the Input of the Draw Node.

ii. Connect Position X to Screen X, Position Y to Screen Y, and Out Scale to Scale.

a. Scale in Draw Texture can be found by clicking the bottom arrow of the node to expand the options.

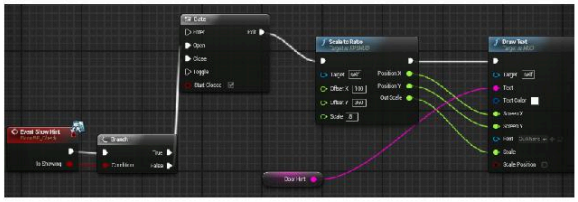
iii. You can now change the values of Offset X, Offset Y, and Scale that will affect the Outputs.

a. Set the Door Hint text so that it is near the center of the screen.

b. Set the Battery so that it is at the lower left corner of the screen.

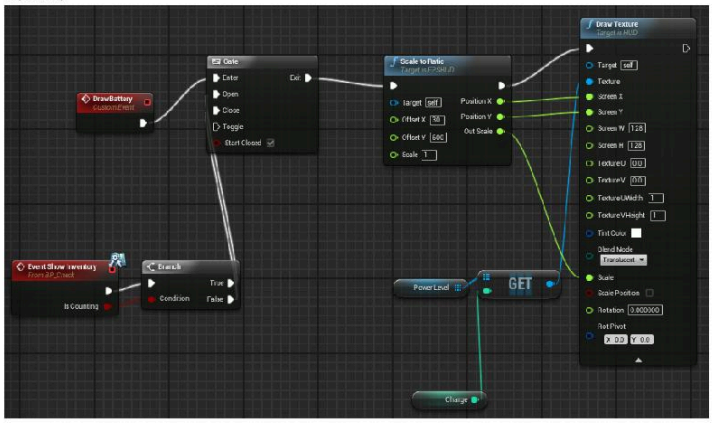
c. Make sure that Scale is not a value of 0. If it is then the draw will not appear.

iv. EX:



4. Right Click and create a Custom Event called DrawHint.
    i. Connect DrawHint to the Enter of Gate for the Draw Text.

5. Right Click and create a Custom Event called DrawBattery.
    i. Connect DrawBattery to the Enter of Gate for the Draw Texture.
    ii. EX:



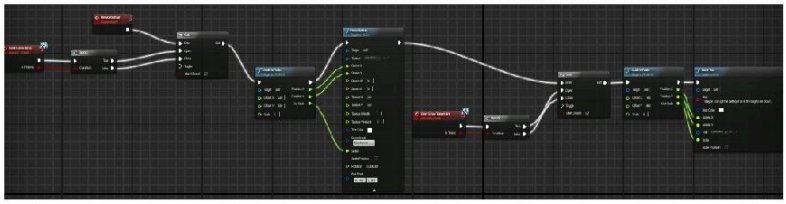
6. Go back to Event Receive Draw HUD and drag to place the Draw Battery Call Function.
    i. Drag the Output of Draw Battery and place the Draw Hint Call Function.
    ii. EX:

7. Compile and Save. On play, the HUD should still appear as it did in the previous exercise.

8. Find an open space and Right Click to create another Custom Event. Name it DrawCrosshair.
    i. Drag the Output and place a Gate.
        a. Right Click and place Event Game Time.
        b. Drag the arrow to place a Branch. Connect Is Playing to the Condition.
        c. Connect True to the Open of Gate and False into Close.
    ii. Drag the Exit of Gate and place the Scale to Ratio Call Function.
    iii. Drag the Output arrow of Scale to Ratio and place Draw Texture.
        a. Connect Position X to Screen X, Position Y to Screen Y, and Out Scale to Scale.
        b. Have Offset X and Offset Y be values that help the player aim. I used a value of 630 for Offset X and 330 for Offset Y. Also make sure Scale is not 0.
    iv. Set the Texture value for Draw Texture to the Crosshair that was imported.

a. Set Screen W and Screen H to a smaller value to give the player an idea of where the coconut will hit. I used a value of 16 for each.
b. Set Texture UWidth and Texture UHeight to 1.

9. Drag the Output of Draw Texture and place a Gate.
i. Right Click and place the Event ShowTargetHint.
a. Drag the Output arrow and place a Branch.
b. Connect Is Trying to Condition.
c. This connection will be **different** from the others as False will be connecting to Open while True will be connecting to Close.
ii. Drag the Exit of Gate and place the Call Function ScaleToRatio.
a. Use a value for Offset X and Offset Y so that the text appears below the crosshair.
b. Make sure Scale is not 0.
iii. Drag the Output arrow of Scale to Ratio and place Draw Text.
a. Set Text to: Maybe I can get the battery if all of the targets are down.
I. A downside to using blueprint draw text is that you are unable to use the newline command in the text. This means that you would have to find a way around it using a variety of nodes. If you are creating a text heavy project, then use coding instead of blueprint.
b. Set Text Color to White.
c. Connect Screen X to Position X and Screen Y to Position Y.
d. Set Font to your imported font.
e. Connect Scale to Out Scale.
iv. EX:



10. Go Back to Event Receive Draw HUD. Drag the Output of Draw Hint and place the Call Function DrawCrosshair.

11. Compile and Save. On Play, the player will be able to see the crosshair and text when standing on the mat.

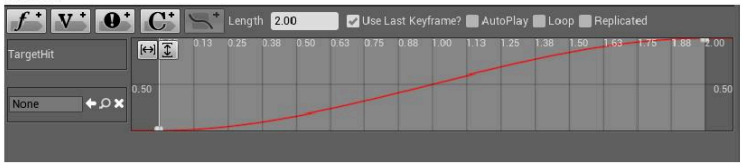
Target

1. Open BP_Target and go to Components.
i. Have the Pole's Collision Presets be set to Custom with everything being Ignore except for Pawn and Projectile.
ii. Have the Target's Collision Presets be set to Custom with everything set to Ignore except for Projectile.

2. Go to Graph and have Target mesh highlighted in My Blueprint.
    i. Right Click and place OnComponentHit for Target.
    ii. Create a new bool variable called bCanCount.
    iii. Drag the Output arrow of OnComponentHit and place a Branch.
        a. Connect a Get of bCanCount to the Condition.
    iv. Drag False of Branch and place Play Sound Attached.
        a. Set Sound to thump_Cue.
        b. Place Get of Target and connect it to Attach to Component.
    v. Drag the Output and place Set bCanCount. Make sure it is checked.

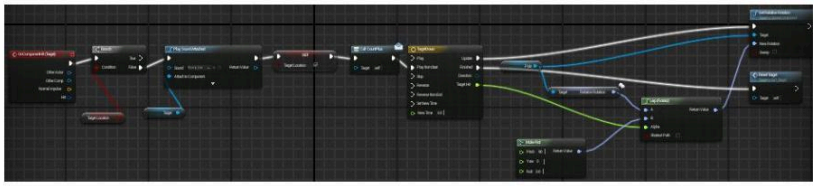
3. Create two Event Dispatchers. One called CountPlus and the other called CountMinus.
    i. Drag the Output of Set CanCount and place Call CountPlus.

4. Right Click and add a Timeline called TargetDown. Double click to open the TargetDown.
    i. Add a float track called TargetHit.
    ii. Have Index 0 be 0 and 0 while Index 1 is 2 and 1.
        a. Remember that Shift + Left Click creates a key.
    iii. Select both key frames, Index 0 and 1, and Right Click to select Cubic Auto.
    iv. Make sure Use Last Keyframe is checked and that Length is 2.
    v. EX:



5. Go back to Event Graph and connect the Output of Call CountPlus to Play from Start.
    i. Drag Update and place Set Actor Relative Rotation.
    ii. Place the Get of the Pole Mesh and connect it to Target.
    iii. Right Click and place Lerp(Rotator).
        a. Drag the Output of Get Pole and place Get Relative Rotation.
        b. Connect Relative Rotation to A of Lerp.
        c. Right Click and place Make Rot. Set Pitch to 90 and connect the Return Value to B.
        d. Connect Target Hit of TargetDown to Alpha of Lerp.
        e. Connect the Return Value of Lerp to New Rotation.

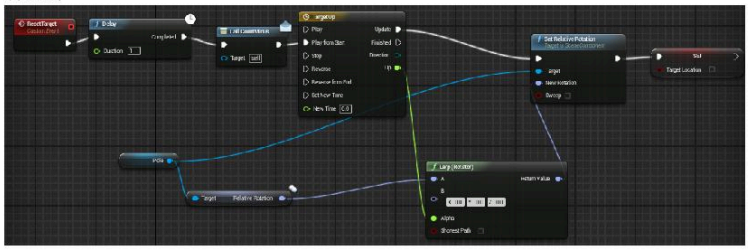
6. Go back towards the beginning with OnComponentHit. Create a Custom Event called ResetTarget.
    i. Drag the Update of TargetDown and place the Call Function of ResetTarget.
    ii. EX:



7. Drag the Output of the ResetTarget Custom Event and place Delay. Make the Delay a value that will not make the game too hard. A hard value would be 0.1 while an easy value would be 2.
    i. Drag the Output of Delay and place Call CountMinus.

8. Right Click and Add Timeline called TargetUp. Double Click to open.
    i. Create a Float Track called TargetRise.
    ii. Follow the same instructions as 4ii to 4iv.

9. Go back to the Event Graph.
    i. Connect the Output of Call CountMinus to Play from Start of TargetUp.
    ii. Drag Update and place Set Actor Relative Location.
        a. Drag the Output and place Set bCanHit. Make sure bCanHit is unchecked.
    iii. Right Click and place Lerp(Rotator).
        a. Connect TargetRise to Alpha.
        b. Place a Get of Pole.
        c. Drag Pole into Target of Set Relative Rotation.
        c. Drag Pole and place Get Relative Location. Drag its Output to A.
    iv. Drag the Return Value of Lerp into the New Rotation of Set Relative Location.
    v. EX:



10. Compile and Save. On play, you will be able to hit the targets down and have them rise.

Count

1. Open the Level Blueprint.
    i. Go to the For Loop and change the Last Index to 2, or a value that would lead to 3 batteries.

2. Select the 3 targets in the scene and Right Click to Add references of selected actor.

i. Drag the Output of BP_Target and place Bind Event to CountPlus.
    a. Connect the other BP_Target's to Target.
    b. Select all *BP_Target's* and Right Click to select Collapse Node. Name it TargetGroup.
ii. Drag Event of the Bind and create a Custom Event called HitTarget.
    a. Drag the Input and connect it to the Output of Bind Event to OnPickUp.
    b. Drag the Output and create a Branch.
    c. Connect the Get of bHasWon into the Condition.
iii. Create an int called DownCount.
    a. Place a Get of DownCount on the graph.
    b. Drag the Output and place Integer + Integer. Have 1 in the empty slot.
    c. Drag the Output and place a Set DownCount. Connect the Input to False of Branch.
iv. Drag the Output of Set DownCount and place a Branch.
    a. Drag the Condition and place Integer >= Integer.
    b. Place a Get of DownCount and connect it to A of Integer >= Integer. Have the bottom slot be 3.
v. Drag the True of Branch and place Set bHasWon.
    a. Make sure that bHasWon is checked.
vi. Place a Reference to Shack Music and drag to place Stop.
    a. Connect the Output of Set to the Input of Stop.

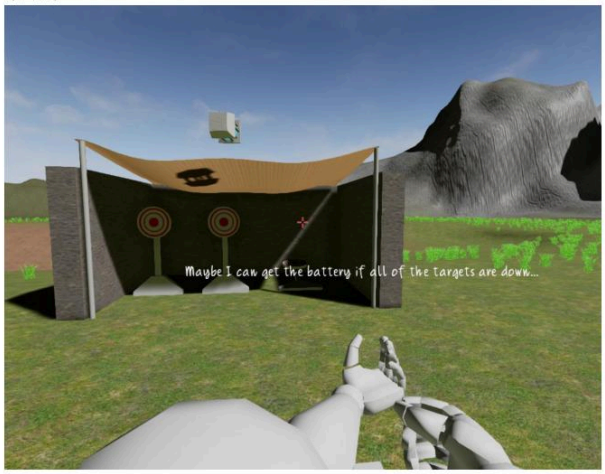
3. In the scene, select the WinBattery. Go back to the Level Blueprint and add a Reference.
i. Drag the Output and place Get Actor Location.
    a. Right Click and place Play Sound at Location.
    b. Set Sound to Win Music.
    c. Connect Return Value to Location.
ii. Select BP_Battery and place a Reference.
    a. Drag the Output of Set Play Sound at Location and place Destroy Actor.
    b. Connect the BP_Battery reference to Target.
iii. Drag the Output of Destroy Actor and place Spawn Actor from Class.
    a. Change Class to BP_Battery_C.

4. Select the WinBattery node and place Get Actor Transform.
i. Drag the Output and place Get Actor Transform.
ii. Connect the Return Value to Spawn Transform.
iii. Drag the Return Value of SpawnActor and place Bind Event to OnPickUp.
    a. Connect Event to the custom event that the other Bind Event to OnPickUp is connected.

5. Create a copy of TargetGroup and drag the Outputs to create Bind Event to CountMinus.
i. Connect the Input of Bind Event to CountMinus to the Output of Bind Event to CountPlus.
ii. Drag Event and create the Custom Event called TargetClimb.
    a. Drag the Output and place a Branch.
    b. Place a Get of bHasWon and connect it to the Condition.

c. Place a Get and Set of DownCount.
d. Drag the Output of Get and place Integer – Integer. Set 1 to the open slot.
e. Drag its Output into DownCount of Set.
f. Connect the Input of Set DownCount into False of Branch.

6. Compile and Save. The player will now be able to win the game and collect the battery.
   i. EX:



Result

The player should be able to shoot projects at the targets. Knocking down targets will give the player points that will unlock the battery.

Submission

Packaged projects will be checked in class on Monday (11/17/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Check for updates to see if they have fixed newline in Blueprint:
https://www.unrealengine.com/

CGT 345

Due Monday (11/17/14)

Goal

Shoot down the targets and collect the prize.

Setup

1. Create a copy of you previous project.

2. Import the two audio files and find two files for Shack Music and Win Music.
    i. Again if you would like to find your own music search here: http://www.freesound.org/

3. Import the crosshair.tga.

Sound

1. Import the Audio files into Sounds.
    i. Double Click the Shack Music.
        a. Under Compression make sure Looping is checked.
    i. Right Click on Swoosh and select Create Cue.
        a. Drag the Output of Wave Player and place a Modulator.
        b. Set Pitch Min to 0.6 and Pitch Max to 1.5.
        c. Drag the Output of Modulator into Output.
    ii. Right Click on thump and select Create Cue.
        a. Drag the Output of Wave Player and place a Modulator.
        b. Set Pitch Min to 0.7 and Pitch Max to 1.5.
        c. Drag the Output of Modulator into Output.

2. Drag the Shack Music onto the scene.
    i. Make sure Allow Spatialization is unchecked under Attenuation.
    ii. Make sure Auto Activate is unchecked under Activation.
    iii. Make sure Volume Multiplier is .2.

Projectile

1. Create a blueprint actor called BP_Projectile.
    i. Open BP_Projectile and go to Components.

ii. Create a Sphere component called CollisionComp.
a. Set the Sphere Radius to 15.0 or whatever radius that fits your projectile.
iii. Create a Projectile Movement component.
a. Set Initial Speed to 3000.
b. Set Max Speed to 3000.
c. Make sure Rotation Follows Velocity is Checked.
d. Make sure that Should Bounce is Checked.
e. Set Bounciness to 0.3.
iv. Create a Static Mesh component called ProjectileMesh.
a. Set Static Mesh to the Coconut mesh.
v. Go to the Defaults tab.
a. Search for Initial Life Span and set it to 3.0.
vi. Compile and Save.

2. In your project folder, go to the Project\Config.
i. Open DefaultEngine.ini and add the following:

```
[/Script/Engine.CollisionProfile]
   +DefaultChannelResponses=(Channel=ECC_GameTraceChannel1, Name=Projectile)
+Profiles=(Name="Projectile",
CollisionEnabled=QueryOnly,ObjectTypeName=Projectile, CustomResponses=( \
   (Channel=Static,             Response=ECR_Block), \
   (Channel=PawnMovement,       Response=ECR_Block), \
   (Channel=Dynamic,            Response=ECR_Block), \
   (Channel=PhysicsBody,        Response=ECR_Block), \
   (Channel=VehicleMovement,    Response=ECR_Block), \
   (Channel=Destructible,       Response=ECR_Block) \
   ))
```

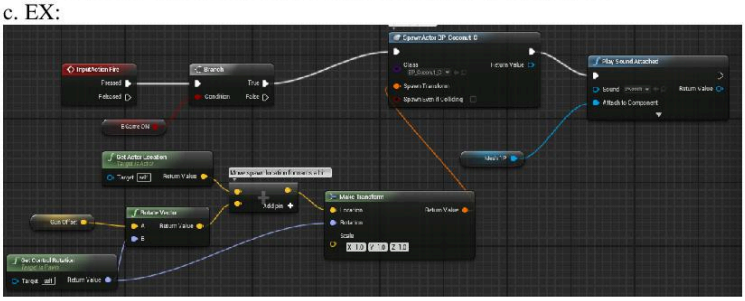
ii. Save and open back up BP_Projectile.
a. Select CollisionComp and go down to Collision.
b. Set the Collision Preset to the Projectile created above.
iii. Compile and Save.
iv. OPTIONAL: Open BP_Projectile and go to Graph.
a. Right Click and place Event Hit.
b. Search Find a Node and place Add Impulse at Location.
c. Drag Other Comp into Target.
d. Place ProjectileMovement onto the graph.
e. Drag and place a Get of Velocity.
f. Drag Velocity and place Vector * Float. Set the float value to 100.
g. Drag the Output into Impulse.
h. Drag Hit Location into Location.
i. Compile and Save.

3. Open BP_FPSCharacter and go to Graph.
i. Create a bool called gameOn and a Vector called MuzzleOffset.
a. Compile and Save.
b. Set MuzzleOffset to 100, 33, 10.
ii. Right Click and place the Action Event called Fire.
a. Drag Pressed and place a Branch.

b. Place a Get of gameOn and attach it to the Condition of Branch.
c. Drag True and place Spawn Actor From Class.
d. Set Class to BP_Projectile.

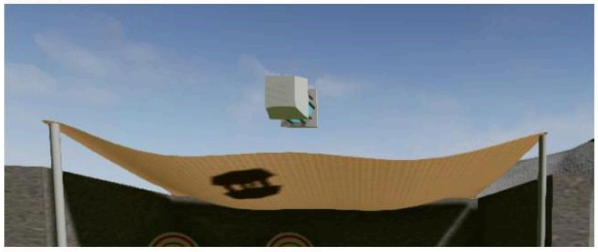
iii. Right Click and place Get Control Rotation.
a. Drag the Return Value and place Rotate Vector.
b. Place a Get of MuzzleOffset and connect it to A of Rotate Vector.
c. Drag the Return Value of Rotate Vector and place Vector + Vector.
d. Right Click and place Get Actor Location and drag its Return Value into Vector + Vector.

iv. Drag the Return Value of Vector + Vector and place Make Transform.
a. Drag the Return Value of Get Control Rotation into Rotation of Make Transform.
b. Drag the Return Value of Make Transform into the Spawn Transform of SpawnActor.
b. Drag Output arrow of SpawnActor and place Play Sound Attached.
A. Set Sound to swoosh_Cue.
B. Place a Get of Mesh and connect it Attach to Component.
c. EX:

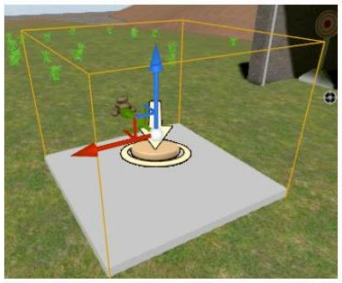


Shooting Zone

1. In the scene, place the battery blueprint above the shooting range.
i. EX:



2. Create a box trigger and have it cover the mat mesh. Name it ShackStart.
i. You can drag ShackStart onto the mat mesh in order to parent it in the Scene Outliner.
ii. EX:



iii. In between ShackStart and BP_Shack, place a target point called WinBattery.
    a. Make Sure it is above the terrain as we will be using this to spawn the battery.

3. Open BP_Check.
    i. Create a new function called GameTime.
        a. Add a bool to Inputs called bIsPlaying.
    ii. Create a new function called ShowTargetHint.
        a. Add a bool to Inputs called bIsTrying.
    iii. Compile and Save.
        a. Make sure to also save in the Content Browser as blueprints connected to BP_Check will change.

4. Make sure that ShackStart is selected in the scene and open the Level Blueprint.
    i. Right Click and add OnActorBeginOverlap for ShackStart.
        a. Drag and place the Interface Message called GameTime.
        b. Right Click and place Get Player Pawn. Connect the Return Value to Target.
        c. Make sure Is Playing is checked.
    ii. Drag and place another GameTime Interface Message.
        a. Right Click and place Get Player Controller.
        b. Drag the Return Value and place Get HUD.
        c. Drag the Return Value into Target.
        d. Make Sure the Is Playing is checked.
    iii. Drag and place the Interface Message of Show Target Hint.
        a. Right Click and place Get Player Controller.
        b. Drag the Return Value and place Get HUD.
        c. Drag Return Value into Target.
        d. Create a bool called bHasWon. Place a Get of the variable and connect it to Is Trying.
    iv. Select the Shack Music in the scene and place a Reference in the Level Blueprint.
        i. Drag the Output of Show Target Hint and place a Branch.
            a. Place a Get of bHasWon and connect it to Condition.
        ii. Drag the Output of Shack Music and place Play.
        iii. Connect False to the Input of Play.
    v. Right Click and place an OnActorEndOverlap for ShackStart.
        a. Place another Reference to Shack Music and drag its Output to place Stop.
        a. Place another two Interface Messages for GameTime and create a chain.

b. Make sure one Targets the Pawn while the other Targets the HUD.
c. Make is Is Playing is unchecked for both.

5. Compile and open BP_FPSCharacter.
i. Go to Graph and select Blueprint Props.
a. Under Interfaces add BP_Check_C.
ii. Right Click and place Event Game Time.
a. Drag the output and place Set gameOn, from our code.
b. Connect Is Playing to the gameOn of Set.

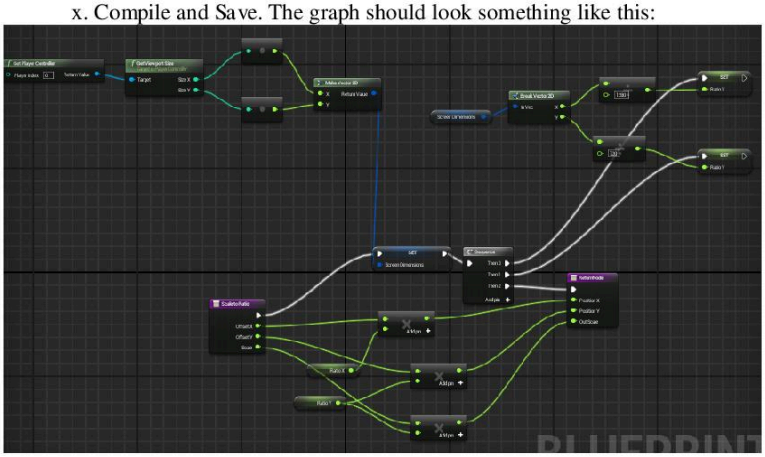
6. Compile and Save. On play, the character will only be able to fire projectiles on the mat.

HUD

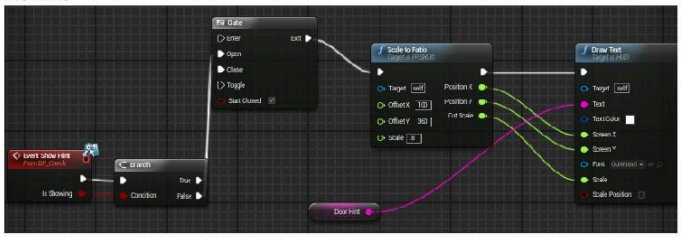
1. Open the FPSHUD blueprint. We will now make some changes to straighten the graph.

2. First create a new Function and call it ScaleToRatio. Double Click it in My Blueprint to open its graph.
i. Create 3 floats under Inputs called OffsetX, OffsetY, and Scale.
ii. Create 3 floats under Outputs called PositionX, PositionY, and OutScale.
iii. Go to a section top left of the Scale to Ratio node. Right Click and place Get Player Controller.
a. Drag the Return Value and place Get Viewport Size.
iv. Right Click and place Make Vector 2D.
a. Connect Size X and Size Y to the X and Y of Make Vector 2D.
v. Right Click the Return Value of Make Vector 2D and Promote to Variable. Name it ScreenDimensions.
vi. Drag Set ScreenDimensions down and drag the Output arrow of Scale to Ratio into the Input arrow.
vii. Drag the Output of Set ScreenDimensions and place a Sequence. Make sure the Output goes to Then 2.
a. Above Sequence place a Get of ScreenDimensions.
b. Drag the Output and place Break Vector 2D.
c. Use a Float/Float node for X and Y. Divide X by 1280 and Y by 720.
d. Create 2 float variables called RatioX and RatioY.
e. Create a Set of each and have the X divide Output connect to RatioX while the other to RatioY.
viii. Connect the Then 0 of Sequence to the Input of Set RatioX, Then 1 to the Input of Set RatioY, and Then 2 to the Input arrow of ReturnNode.
ix. Connect OffsetX, OffsetY, and Scale of Scale to Ratio to each of their own Float*Float nodes.
a. Place a Get of RatioX and connect it to the Float*Float of OffsetX. Connect the Output to PositionX of ReturnNode.
b. Place a Get of RatioY and connect it to the Float*Float of OffsetY and Scale. Drag the Output from OffsetY multiplier into PositionY and the Output from the Scale multiplier into OutScale.

x. Compile and Save. The graph should look something like this:



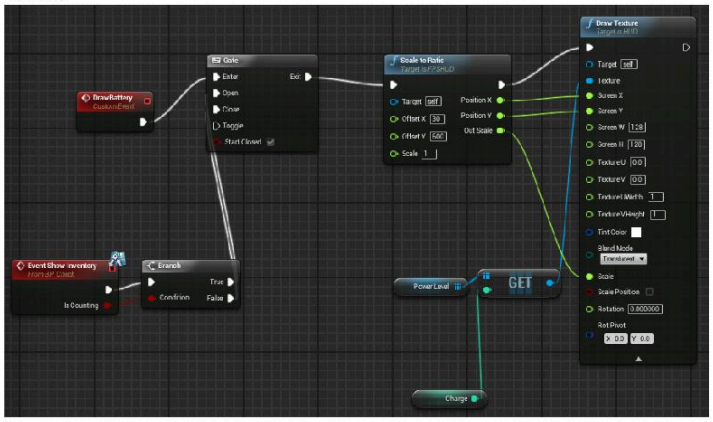
3. Go back to Event Graph.
    i. Disconnect from the Exit of Gate from each. Place the function ScaleToRatio between the Gate and Draw node.
        a. Connect the Exit of Gate into the Input of Scale to Ratio. Connect the Output to the Input of the Draw Node.
    ii. Connect Position X to Screen X, Position Y to Screen Y, and Out Scale to Scale.
        a. Scale in Draw Texture can be found by clicking the bottom arrow of the node to expand the options.
    iii. You can now change the values of Offset X, Offset Y, and Scale that will affect the Outputs.
        a. Set the Door Hint text so that it is near the center of the screen.
        b. Set the Battery so that it is at the lower left corner of the screen.
        c. Make sure that Scale is not a value of 0. If it is then the draw will not appear.
    iv. EX:



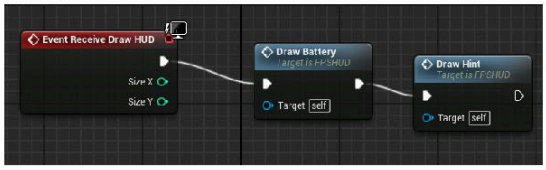
4. Right Click and create a Custom Event called DrawHint.
    i. Connect DrawHint to the Enter of Gate for the Draw Text.

5. Right Click and create a Custom Event called DrawBattery.
    i. Connect DrawBattery to the Enter of Gate for the Draw Texture.

ii. EX:



6. Go back to Event Receive Draw HUD and drag to place the Draw Battery Call Function.
   i. Drag the Output of Draw Battery and place the Draw Hint Call Function.
   ii. EX:

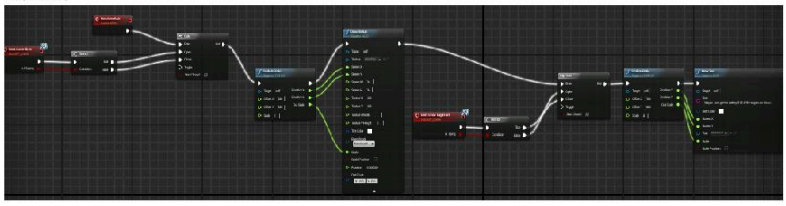


7. Compile and Save. On play, the HUD should still appear as it did in the previous exercise.

8. Find an open space and Right Click to create another Custom Event. Name it DrawCrosshair.
   i. Drag the Output and place a Gate.
      a. Right Click and place Event Game Time.
      b. Drag the arrow to place a Branch. Connect Is Playing to the Condition.
      c. Connect True to the Open of Gate and False into Close.
   ii. Drag the Exit of Gate and place the Scale to Ratio Call Function.
   iii. Drag the Output arrow of Scale to Ratio and place Draw Texture.
      a. Connect Position X to Screen X, Position Y to Screen Y, and Out Scale to Scale.
      b. Have Offset X and Offset Y be values that help the player aim. I used a value of 630 for Offset X and 330 for Offset Y. Also make sure Scale is not 0.
   iv. Set the Texture value for Draw Texture to the Crosshair that was imported.
      a. Set Screen W and Screen H to a smaller value to give the player an idea of where the coconut will hit. I used a value of 16 for each.
      b. Set Texture UWidth and Texture UHeight to 1.

9. Drag the Output of Draw Texture and place a Gate.

i. Right Click and place the Event ShowTargetHint.
    a. Drag the Output arrow and place a Branch.
    b. Connect Is Trying to Condition.
    c. This connection will be **different** from the others as False will be connecting to Open while True will be connecting to Close.
ii. Drag the Exit of Gate and place the Call Function ScaleToRatio.
    a. Use a value for Offset X and Offset Y so that the text appears below the crosshair.
    b. Make sure Scale is not 0.
iii. Drag the Output arrow of Scale to Ratio and place Draw Text.
    a. Set Text to: Maybe I can get the battery if all of the targets are down.
        I. A downside to using blueprint draw text is that you are unable to use the newline command in the text. This means that you would have to find a way around it using a variety of nodes. If you are creating a text heavy project, then use coding instead of blueprint.
    b. Set Text Color to White.
    c. Connect Screen X to Position X and Screen Y to Position Y.
    d. Set Font to your imported font.
    e. Connect Scale to Out Scale.
iv. EX:



10. Go Back to Event Receive Draw HUD. Drag the Output of Draw Hint and place the Call Function DrawCrosshair.

11. Compile and Save. On Play, the player will be able to see the crosshair and text when standing on the mat.

Target

1. Open BP_Target and go to Components.
    i. Have the Pole's Collision Presets be set to Custom with everything being Ignore except for Pawn and Projectile.
    ii. Have the Target's Collision Presets be set to Custom with everything set to Ignore except for Projectile.

2. Go to Graph and have Target mesh highlighted in My Blueprint.
    i. Right Click and place OnComponentHit for Target.
    ii. Create a new bool variable called bCanCount.
    iii. Drag the Output arrow of OnComponentHit and place a Branch.

a. Connect a Get of bCanCount to the Condition.

iv. Drag False of Branch and place Play Sound Attached.

a. Set Sound to thump_Cue.

b. Place Get of Target and connect it to Attach to Component.

v. Drag the Output and place Set bCanCount. Make sure it is checked.

3. Create two Event Dispatchers. One called CountPlus and the other called CountMinus.

i. Drag the Output of Set CanCount and place Call CountPlus.

4. Right Click and add a Timeline called TargetDown. Double click to open the TargetDown.

i. Add a float track called TargetHit.

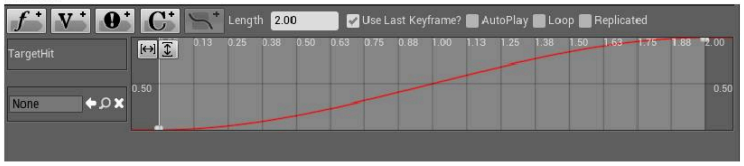
ii. Have Index 0 be 0 and 0 while Index 1 is 2 and 1.

a. Remember that Shift + Left Click creates a key.

iii. Select both key frames, Index 0 and 1, and Right Click to select Cubic Auto.

iv. Make sure Use Last Keyframe is checked and that Length is 2.

v. EX:



5. Go back to Event Graph and connect the Output of Call CountPlus to Play from Start.

i. Drag Update and place Set Actor Relative Rotation.

ii. Place the Get of the Pole Mesh and connect it to Target.

iii. Right Click and place Lerp(Rotator).

a. Drag the Output of Get Pole and place Get Relative Rotation.

b. Connect Relative Rotation to A of Lerp.

c. Right Click and place Make Rot. Set Pitch to 90 and connect the Return Value to B.

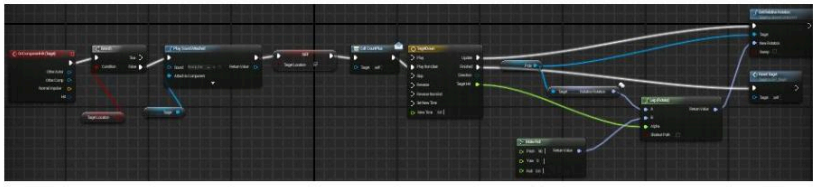
d. Connect Target Hit of TargetDown to Alpha of Lerp.

e. Connect the Return Value of Lerp to New Rotation.

6. Go back towards the beginning with OnComponentHit. Create a Custom Event called ResetTarget.

i. Drag the Update of TargetDown and place the Call Function of ResetTarget.

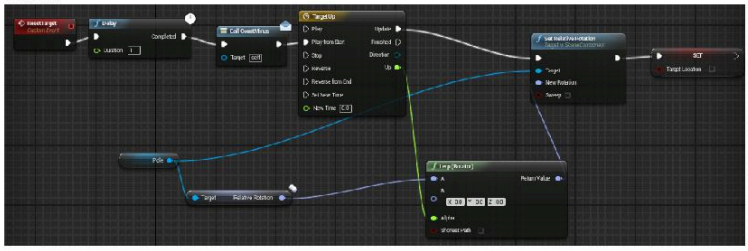
ii. EX:



7. Drag the Output of the ResetTarget Custom Event and place Delay. Make the Delay a value that will not make the game too hard. A hard value would be 0.1 while an easy value would be 2.
    i. Drag the Output of Delay and place Call CountMinus.

8. Right Click and Add Timeline called TargetUp. Double Click to open.
    i. Create a Float Track called TargetRise.
    ii. Follow the same instructions as 4ii to 4iv.

9. Go back to the Event Graph.
    i. Connect the Output of Call CountMinus to Play from Start of TargetUp.
    ii. Drag Update and place Set Actor Relative Location.
        a. Drag the Output and place Set bCanHit. Make sure bCanHit is unchecked.
    iii. Right Click and place Lerp(Rotator).
        a. Connect TargetRise to Alpha.
        b. Place a Get of Pole.
        c. Drag Pole into Target of Set Relative Rotation.
        c. Drag Pole and place Get Relative Location. Drag its Output to A.
    iv. Drag the Return Value of Lerp into the New Rotation of Set Relative Location.
    v. EX:



10. Compile and Save. On play, you will be able to hit the targets down and have them rise.

Count

1. Open the Level Blueprint.
    i. Go to the For Loop and change the Last Index to 2, or a value that would lead to 3 batteries.

2. Select the 3 targets in the scene and Right Click to Add references of selected actor.
    i. Drag the Output of BP_Target and place Bind Event to CountPlus.
        a. Connect the other BP_Target's to Target.
        b. Select all *BP_Target's* and Right Click to select Collapse Node. Name it TargetGroup.
    ii. Drag Event of the Bind and create a Custom Event called HitTarget.
        a. Drag the Input and connect it to the Output of Bind Event to OnPickUp.
        b. Drag the Output and create a Branch.
        c. Connect the Get of bHasWon into the Condition.

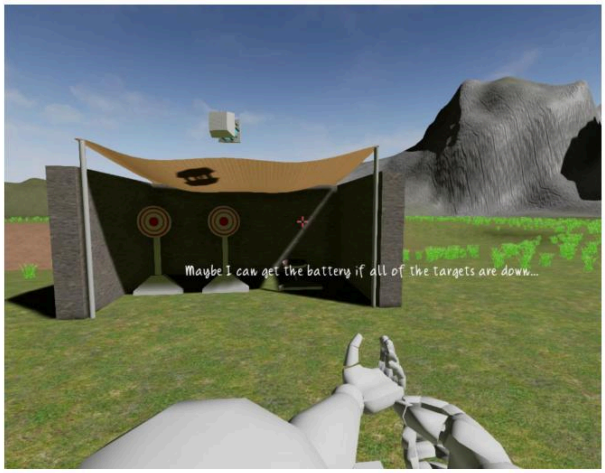
iii. Create an int called DownCount.
    a. Place a Get of DownCount on the graph.
    b. Drag the Output and place Integer + Integer. Have 1 in the empty slot.
    c. Drag the Output and place a Set DownCount. Connect the Input to False of Branch.
iv. Drag the Output of Set DownCount and place a Branch.
    a. Drag the Condition and place Integer >= Integer.
    b. Place a Get of DownCount and connect it to A of Integer >= Integer. Have the bottom slot be 3.
v. Drag the True of Branch and place Set bHasWon.
    a. Make sure that bHasWon is checked.
vi. Place a Reference to Shack Music and drag to place Stop.
    a. Connect the Output of Set to the Input of Stop.

3. In the scene, select the WinBattery. Go back to the Level Blueprint and add a Reference.
i. Drag the Output and place Get Actor Location.
    a. Right Click and place Play Sound at Location.
    b. Set Sound to Win Music.
    c. Connect Return Value to Location.
ii. Select the battery above the shooting range and place a Reference.
    a. Drag the Output of Set Play Sound at Location and place Destroy Actor.
    b. Connect the reference to Target.
iii. Drag the Output of Destroy Actor and place Spawn Actor from Class.
    a. Change Class to BP_Battery_C.

4. Select the WinBattery node and place Get Actor Transform.
i. Drag the Output and place Get Actor Transform.
ii. Connect the Return Value to Spawn Transform.
iii. Drag the Return Value of SpawnActor and place Bind Event to OnPickUp.
    a. Connect Event to the custom event that the other Bind Event to OnPickUp is connected.

5. Create a copy of TargetGroup and drag the Outputs to create Bind Event to CountMinus.
i. Connect the Input of Bind Event to CountMinus to the Output of Bind Event to CountPlus.
ii. Drag Event and create the Custom Event called TargetClimb.
    a. Drag the Output and place a Branch.
    b. Place a Get of bHasWon and connect it to the Condition.
    c. Place a Get and Set of DownCount.
    d. Drag the Output of Get and place Integer – Integer. Set 1 to the open slot.
    e. Drag its Output into DownCount of Set.
    f. Connect the Input of Set DownCount into False of Branch.

6. Compile and Save. The player will now be able to win the game and collect the battery.
i. EX:



Result

The player should be able to shoot projects at the targets. Knocking down targets will give the player points that will unlock the battery.

Submission

Packaged projects will be checked in class on Monday (11/17/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Check for updates to see if they have fixed newline in Blueprint:
https://www.unrealengine.com/

CGT 345

Due Monday (12/1/14)

Goal

Create a start menu that takes the player to the main game, instructs on the goals, and allows them to quit from the menu.

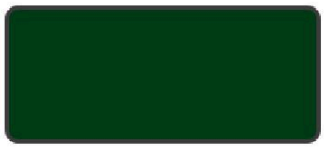
Setup

1. Create a copy of you previous project.

Menu Assets

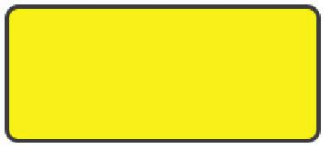
1. Create the following images for the Main Menu. Make sure they are saved as a PNG for transparency.
    i. A logo for the game Survival Island. It should have the name of the game as well as a graphical image.
        a. Make it 256x256.
        b. EX:

    ii. 3 buttons that are 256x168:
        a. Default:



b. Hover:

c. Active:



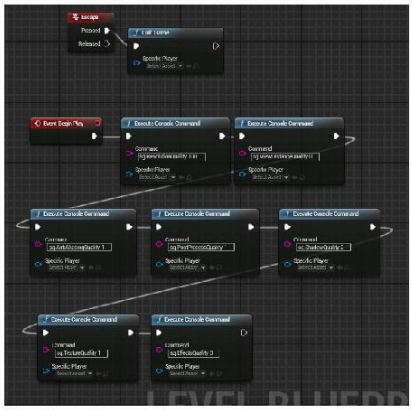
2. Import a click sound into the Sounds folder for menu presses.

Scene Setup

1. Go into your Island folder.
    i. Look for the Level.umap file.
        a. If you saved it in a Maps folder then it should be under Island > Content > Maps.
    ii. Create a copy of the map and name it MainMenu.

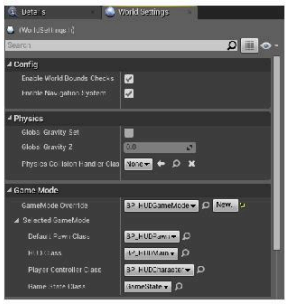
2. Open the project in the Unreal Editor.
    i. Go to Edit > Project Settings.
    ii. Under Game select the Maps & Modes section.
    iii. In Default Maps, change the Game Default Map to MainMenu.
    iv. Press Set as Default.
    v. Go File > Open Level and select MainMenu.

3. Open the Level Blueprint.
    i. Delete everything that is not the game settings and escape option.
    ii. EX:



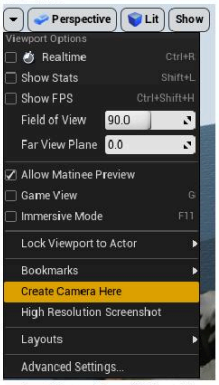
iii. Compile and Save.

4. Go to the Blueprints folder and create the following blueprints.
    i. Create blueprint using Pawn as the parent class called BP_MainPawn.
        a. Open the blueprint to make sure that Use Controller Rotation Pitch, Yaw, and Roll are unchecked in Defaults.
        b. Compile and Save.
    ii. Create blueprint using Player Controller as the parent class called BP_MainChar.
        a. Open and go to Defaults.
        b. Make sure the Input Yaw, Pitch, and Roll Scale are set to 0 under Player Controller.
        c. Go to Mouse Interface and make sure that Show Mouse Cursor, Enable Click Events, and Enable Mouse Over Events are checked.
        d. Compile and Save.
    iii. Create blueprint using HUD as the parent class called BP_MainHUD.
        a. Save.
    iv. Create blueprint using Game Mode as the parent class called BP_MainMenu.
        a. Open and go to Defaults.
        b. Change the Default Pawn Class to BP_MainPawn.
        c. Change the HUD Class to BP_MainHUD.
        d. Change Player Controller Class to BP_MainChar.
        e. Compile and Save.
5. Go back to the main editor and select World Settings above the scene.
    i. Scroll down to Game Mode and change the GameMode Override to BP_MainMenu.
    ii. EX:



iii. Save the level.

6. Pan the perspective camera so that the entire Island is visible.
   i. When in the correct position, select the arrow next to Perspective and click Create Camera Here.
      a. EX:



   ii. Go to the Details of the Camera and uncheck Constrain Aspect Ratio.

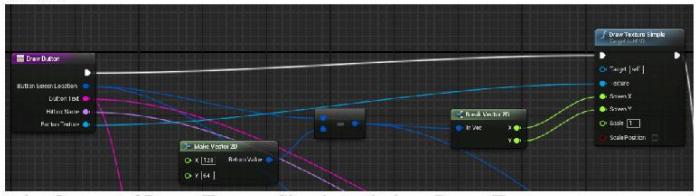
7. Open the Level Blueprint with the Camera Actor selected.
   i. In the Find a Node section search and place Set View Target with Blend.
   ii. Connect its Input arrow into the last Output arrow of the Event Begin Play chain.
   iii. Right Click and place Get Player Controller. Connect its Return Value to Target.
   iv. Right Click and place a reference to your Camera Actor. Connect the Output to New View Target.
   v. Compile and Save.

8. On play, the camera will appear in the selected position with a usable mouse with an unmovable controller.

HUD

1. Open BP_MainHUD and go to Graph.

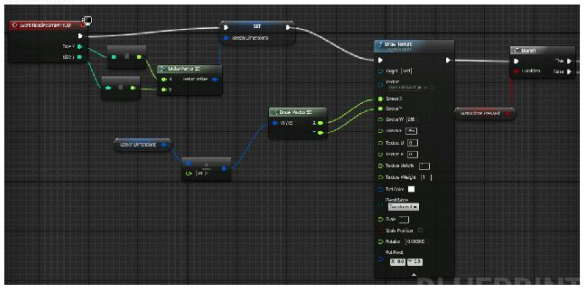
2. Create a new Function called DrawButton and double click to open.
    i. Select the Draw Button node and add the following Inputs.
        a. Vector2D called ButtonScreenLocation.
        b. String called ButtonText.
        c. name called HitboxName.
        d. Texture2D called ButtonTexture.
    ii. Drag the Output arrow of Draw Button and place Draw Texture Simple.
        a. Place a Make Vector 2D
            A. Give it an X of 128 and Y of 64.
        b. Drag the Output and place Vector 2D – Vector 2D.
            A. Connect Button Screen Location of Draw Button to the open slot.
        c. Drag the Output and place Break Vector 2D.
            A. Connect X to Screen X and Y to Screen Y.
        d. EX:



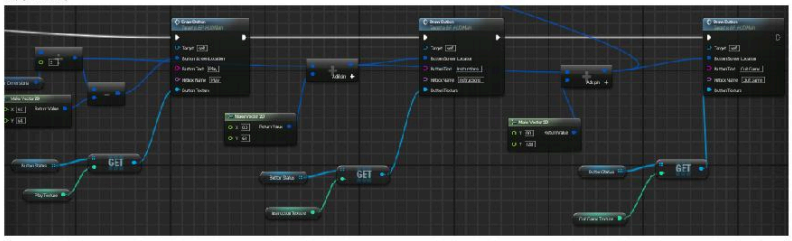
    iii. Drag the Output of Draw Texture Simple and place Draw Text.
        a. Connect Button Text of Draw Button to Text.
        b. Set Text Color to White.
        c. Right Click and place Get Text Size.
            A. Connect Button Text of Draw Button to Text.
            B. Set your imported font.
            C. Set Scale to 1.5.
            D. Place Make Vector 2D and connect Out Width to X and Out Height to Y.
            E. Drag the Return Value to place Vector 2D/ Float. Set the float value to 2.
            F. Drag the Output and place Vector 2D-Vector 2D. Connect ButtonScreenLocation of Draw Button to the open slot.
            G. Drag the Output and place Break Vector 2D.
            H. Connect X to Screen X and Y to Screen Y.
        d. Set Font to your imported font.
        e. Set Scale to 1.5.
    iv. Drag the Output of Draw Text and place Add Hit Box.
        a. Drag the Output of the Vector 2D-Vector 2D of ButtonScreenLocation and Make Vector 2D into Position.
        b. Place a Make Vector 2D with an X of 256 and Y of 128.
            A. Connect the Return Value to Size.
        c. Drag Hitbox Name of Draw Button into Name.

v. Compile and Save.

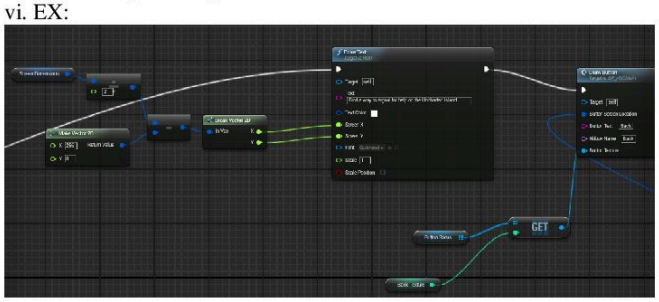
3. Go back to the Event Graph.
   i. Create the following variables.
      a. bool called PlayPressed.
      b. bool called InstructionPressed.
      c. bool called QuitPressed.
      d. bool called BackPressed.
      e. int called PlayTexture.
      f. int called InstructionTexture.
      g. int called QuitTexture.
      h. int called BackTexture.
      i. Vector2D called ScreenDimensions.
      j. Texture2D array called ButtonStatus.
         A. Press the box next to Variable Type to create an array.
         B. Compile and Save.
         C. In Default Value, create 3 elements.
         D. Place Default in 0, Hover in 1, and Active in 2.
   ii. Right Click and place Event Receive Draw HUD.
      a. Drag the Output and place a Set of ScreenDimensions.
         A. Right Click and place a Make Vector 2D. Drag the Return Value into the Screen Dimensions of Set.
         B. Drag Size X into X and Size Y into Y from the Event Receive Draw HUD.
      b. Drag the Output of Set and place Draw Texture.
         A. Set Texture to the title or logo of the game.
         B. Create a Break Vector 2D and connect X and Y to Screen X and Y.
         C. Use the Get of ScreenDimensions and some type of offset as the In Vec for Break Vector 2D.
            1. I Divided Screen Dimensions by a float of 20.
         D. Set Screen W and H to 256.
         E. Set Texture UWidth and VHeight to 1.
         F. Make sure Scale is 1.
      c. Drag the Output of Draw Texture and place a Branch.
         A. Place a Get of InstructionPressed and connect it to Condition.
   iii. EX:



4. Drag the False of Branch and place DrawButton.
    i. Place a Get of ScreenDimensions.
        a. Drag and place Vector2D/Float. Set the Float to 2.
        b. Drag the Output and place Vector2D – Vector2D.
            A. Right Click and place a Make Vector 2D.
            B. Set X to 0 and Y to 64.
            C. Connect the Return Value to the open slot of Vector2D – Vector2D.
        c. Drag the Output into Button Screen Location.
        d. Set Button Text and Hitbox Name to Play.
        e. Place a Get of Button Status.
        f. Drag the Output and place a GET.
            A. Set the int value to the Get PlayTexture.
        g. Connect the Output of GET into Button Texture.
    ii. Drag the Output of Draw Button and place DrawButton.
        a. Drag the Output of the previous Vector2D/Float and place Vector2D + Vector2D.
            A. Right Click and place Make Vector 2D.
            B. Set Y to 64.
            C. Connect the Return Value to Vector2D + Vector2D.
        b. Connect the Output to Button Screen Location.
        c. Set Button Text and Hitbox Name to Instructions.
        d. Place Get of Button Status.
        e. Drag the Output and place a GET.
            A. Set the int value to the Get of InstructionTexture.
        f. Drag the Output into Button Texture.
    iii. Drag the Output of Draw Button and place DrawButton.
        a. Get the Output of the previous Vector2D + Vector2D and place another Vector2D + Vector2D.
            A. Right Click and place a Make Vector 2D.
            B. Set Y to 128.
            C. Connect the Return Value to the open slot of Vector2D + Vector2D.
        b. Drag the Output into Button Screen Location.
        c. Set Button Text and Hitbox Name to Quit Game.
        d. Place Get of Button Status.
        e. Drag the Output and place a GET.
            A. Set the int value to the Get of QuitTexture.
        f. Drag the Output into Button Texture.
    iv. EX:

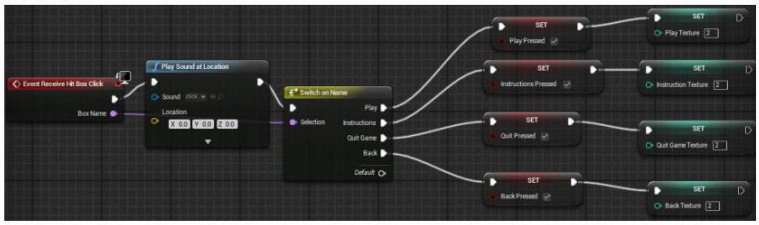


5. Drag the True of Branch and place Draw Text.
    i. Place a Get of ScreenDimensions.
        a. Drag and place Vector2D/Float. Set the Float to 2.
        b. Drag the Output and place Vector2D – Vector2D.
            A. Right Click and place a Make Vector 2D.
            B. Set X to 256 and Y to 0.
            C. Connect the Return Value to the open slot of Vector2D – Vector2D.
        c. Drag the Output and place a Break Vector 2D.
        d. Connect X to Screen X and Y to Screen Y.
    ii. Set Text to: Find a way to signal for help on the Uncharted Island.
    iii. Set Texture Color to White.
    iv. Set your Font.
    v. Drag the Output of Draw Text and place Draw Button.
        a. Find the Vector2D + Vector2D Output used for the Quit Game Button. Connect it to the current Button Screen Location.
        b. Set Button Text and Hitbox Name to Back.
        c. Place Get of Button Status.
        d. Drag the Output and place a GET.
            A. Set the int value to the Get of BackTexture.
        e. Drag the Output into Button Texture.
    vi. EX:

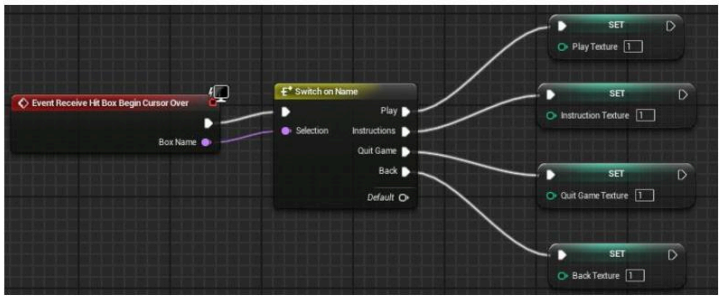


6. Right Click and place Event Receive Hit Box Click.
    i. Drag the Output arrow and place Play Sound at Location.
        a. Set Sound to click.
    i. Search and place Switch on Name in the Find a Node section.
        a. Connect the Output arrow of Click to the Input of Play Sound at Location.
        b. Connect Box Name of Hit Box Click to Selection.
        c. Select Switch and create 4 elements for Pin Names.
            A. 0 is Play, 1 is Instructions, 2 is Quit Game, and 3 is Back.
            B. Make sure the above values are the same as your Hitbox Names from DrawButton.
    ii. Drag Play and place a Set of PlayPressed.
        a. Make sure Play Pressed is checked.
        b. Drag the Output and place a Set of PlayTexture with a value of 2.

iii. Drag Instructions and place a Set of InstructionPressed.
a. Make sure Instruction Pressed is checked.
b. Drag the Output and place a Set of InstructionTexture with a value of 2.
iv. Drag Quit Game and place a Set of QuitPressed.
a. Make sure Quit Pressed is checked.
b. Drag the Output and place a Set of QuitTexture with a value of 2.
v. Drag Back and place a Set of BackPressed.
a. Make sure Back Pressed is checked.
b. Drag the Output and place a Set of BackTexture with a value of 2.
vi. EX:

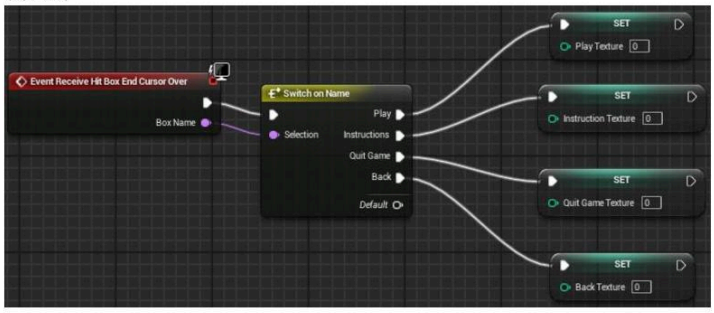


7. Right Click and place Event Receive Hit Box Begin Cursor Over.
i. Copy and paste the Switch on Name with the proper Outputs.
a. Connect the Output arrow of Click to the Input of Switch.
b. Connect Box Name to Selection.
ii. Drag Play and place a Set of PlayTexture with a value of 1.
iii. Drag Instructions and place a Set of InstructionTexture with a value of 1.
iv. Drag Quit Game and place a Set of QuitTexture with a value of 1.
v. Drag Back and place a Set of BackTexture with a value of 1.
vi. EX:

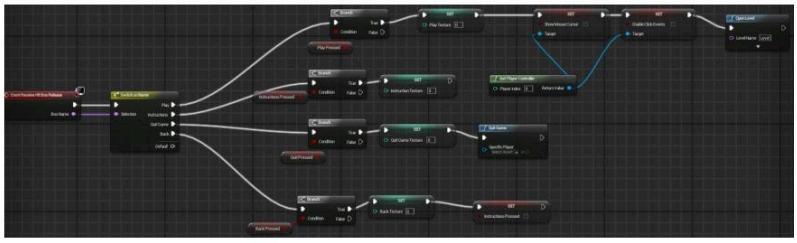


8. Right Click and place Event Receive Hit Box End Cursor Over.
i. Copy and paste the Switch on Name with the proper Outputs.
a. Connect the Output arrow of Click to the Input of Switch.
b. Connect Box Name to Selection.
ii. Drag Play and place a Set of PlayTexture with a value of 0.
iii. Drag Instructions and place a Set of InstructionTexture with a value of 0.

iv. Drag Quit Game and place a Set of QuitTexture with a value of 0.
v. Drag Back and place a Set of BackTexture with a value of 0.
vi. EX:



8. Right Click and place Event Receive Hit Box Release.
    i. Copy and paste the Switch on Name with the proper Outputs.
        a. Connect the Output arrow of Click to the Input of Switch.
        b. Connect Box Name to Selection.
    ii. Drag Play and place a Branch.
        a. Place a Get of PlayPressed and connect it to the Condition.
        b. Drag True and place a Set of PlayTexture with a value of 0.
        c. Right Click and place Get Player Controller.
            A. Drag the Return Value and place a Set of Show Mouse Cursor.
            B. Make sure it is unchecked.
            C. Drag the Return Value and place a Set of Enable Click Events.
            D. Make sure it is unchecked.
        d. Connect the Output of Set PlayTexture to the Input of Set Show Mouse Cursor.
        e. Connect the Output of Set Show Mouse Cursor to the Input of Set Enable Click Events.
        f. Drag the Output and place Open Level.
            A. Set Level Name to the name of the previous levels umap file.
    iii. Drag Instructions and place a Branch.
        a. Place a Get of InstructionPressed and connect it to the Condition.
        b. Drag True and place a Set of InstructionTexture with a value of 0.
    iv. Drag Quit Game and place a Branch.
        a. Place a Get of QuitPressed and connect it to the Condition.
        b. Drag True and place a Set of QuitTexture with a value of 0.
        f. Drag the Output and place Quit Game.
    v. Drag Back and place a Branch.
        a. Place a Get of BackPressed and connect it to the Condition.
        b. Drag True and place a Set of BackTexture with a value of 0.
        c. Drag the Output and place a Set of InstructionPressed.
            A. Make sure it is unchecked.
    vi. EX:



9. Compile and Save. On play, you should now see the menu along with the texture changes. Play should take you to the next level, Instructions should show the explanation and Back, Back should take the player back to the previous screen, and Quit Game should exit from the game.

Result

The player should see a working menu that hints to gameplay and gives them the option to play or leave.

Submission

Packaged projects will be checked in class on Monday (12/1/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. A HUD tutorial (Parts were used for the exercise):
Part 1: https://www.youtube.com/watch?v=7gwgU0UPENA
Part 2: https://www.youtube.com/watch?v=s423ydn3t_E

2. A Menu tutorial using meshes: https://www.youtube.com/watch?v=vBADl-gbITQ

3. You can also install this plug in for UI:
Download: http://coherent-labs.com/download-unreal-engine-4/
First in tutorial series: https://www.youtube.com/watch?v=M-WhfJWEgJo

CGT 345

Due Monday (12/1/14)

Goal

Create a start menu that takes the player to the main game, instructs on the goals, and allows them to quit from the menu.

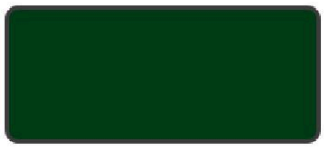
Setup

1. Create a copy of you previous project.

Menu Assets

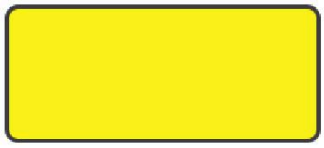
1. Create the following images for the Main Menu. Make sure they are saved as a PNG for transparency.
    i. A logo for the game Survival Island. It should have the name of the game as well as a graphical image.
        a. Make it 256x256.
        b. EX:

    ii. 3 buttons that are 256x168:
        a. Default:



b. Hover:

c. Active:



2. Import a click sound into the Sounds folder for menu presses.

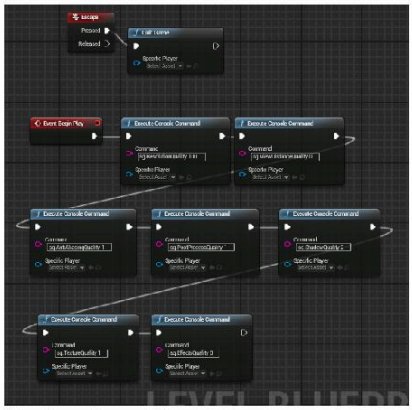
Scene Setup

1. Go into your Island folder.
    i. Look for the Level.umap file.
        a. If you saved it in a Maps folder then it should be under Island > Content > Maps.
    ii. Create a copy of the map and name it MainMenu.

2. Open the project folder on go into the Config folder and open DefaultEngine.ini.
    i. Change the end of the GameDefaultMap to MainMenu.
    ii. Change the end of EditorStartupMap to MainMenu.
    iii. Save.
    iv. EX:

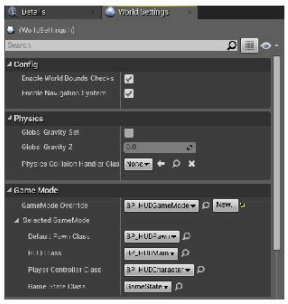
```
[/Script/EngineSettings.GameMapsSettings]
GameDefaultMap=/Game/Maps/MainMenu
EditorStartupMap=/Game/Maps/MainMenu
GlobalDefaultGameMode=/Game/Blueprints/MyGame.MyGame_C
```

3. Open the Unreal Editor and go to the Level Blueprint.
    i. Delete everything that is not the game settings and escape option.
    ii. EX:



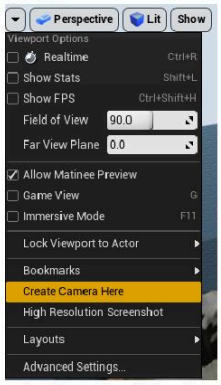
iii. Compile and Save.

4. Go to the Blueprints folder and create the following blueprints.
    i. Create blueprint using Pawn as the parent class called BP_MainPawn.
        a. Open the blueprint to make sure that Use Controller Rotation Pitch, Yaw, and Roll are unchecked in Defaults.
        b. Compile and Save.
    ii. Create blueprint using Player Controller as the parent class called BP_MainChar.
        a. Open and go to Defaults.
        b. Make sure the Input Yaw, Pitch, and Roll Scale are set to 0 under Player Controller.
        c. Go to Mouse Interface and make sure that Show Mouse Cursor, Enable Click Events, and Enable Mouse Over Events are checked.
        d. Compile and Save.
    iii. Create blueprint using HUD as the parent class called BP_MainHUD.
        a. Save.
    iv. Create blueprint using Game Mode as the parent class called BP_MainMenu.
        a. Open and go to Defaults.
        b. Change the Default Pawn Class to BP_MainPawn.
        c. Change the HUD Class to BP_MainHUD.
        d. Change Player Controller Class to BP_MainChar.
        e. Compile and Save.
5. Go back to the main editor and select World Settings above the scene.
    i. Scroll down to Game Mode and change the GameMode Override to BP_MainMenu.
    ii. EX:



iii. Save the level.

6. Pan the perspective camera so that the entire Island is visible.
    i. When in the correct position, select the arrow next to Perspective and click Create Camera Here.
        a. EX:



    ii. Go to the Details of the Camera and uncheck Constrain Aspect Ratio.

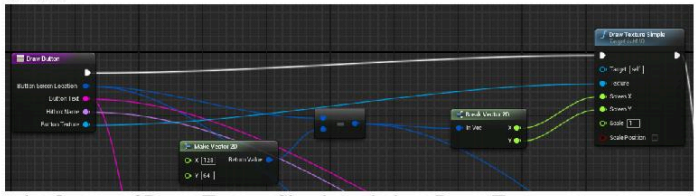
7. Open the Level Blueprint with the Camera Actor selected.
    i. In the Find a Node section search and place Set View Target with Blend.
    ii. Connect its Input arrow into the last Output arrow of the Event Begin Play chain.
    iii. Right Click and place Get Player Controller. Connect its Return Value to Target.
    iv. Right Click and place a reference to your Camera Actor. Connect the Output to New View Target.
    v. Compile and Save.

8. On play, the camera will appear in the selected position with a usable mouse with an unmovable controller.

HUD

1. Open BP_MainHUD and go to Graph.

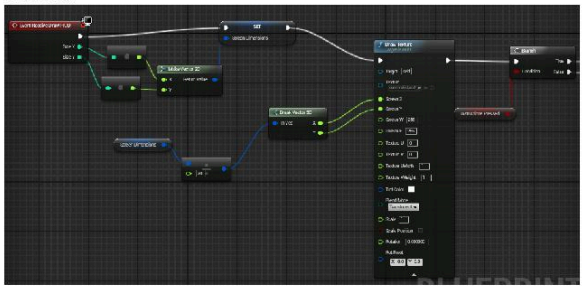
2. Create a new Function called DrawButton and double click to open.
    i. Select the Draw Button node and add the following Inputs.
        a. Vector2D called ButtonScreenLocation.
        b. String called ButtonText.
        c. name called HitboxName.
        d. Texture2D called ButtonTexture.
    ii. Drag the Output arrow of Draw Button and place Draw Texture Simple.
        a. Place a Make Vector 2D
            A. Give it an X of 128 and Y of 64.
        b. Drag the Output and place Vector 2D – Vector 2D.
            A. Connect Button Screen Location of Draw Button to the open slot.
        c. Drag the Output and place Break Vector 2D.
            A. Connect X to Screen X and Y to Screen Y.
        d. EX:



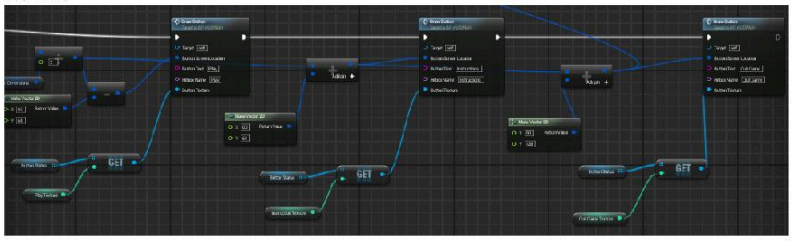
    iii. Drag the Output of Draw Texture Simple and place Draw Text.
        a. Connect Button Text of Draw Button to Text.
        b. Set Text Color to White.
        c. Right Click and place Get Text Size.
            A. Connect Button Text of Draw Button to Text.
            B. Set your imported font.
            C. Set Scale to 1.5.
            D. Place Make Vector 2D and connect Out Width to X and Out Height to Y.
            E. Drag the Return Value to place Vector 2D/ Float. Set the float value to 2.
            F. Drag the Output and place Vector 2D-Vector 2D. Connect ButtonScreenLocation of Draw Button to the open slot.
            G. Drag the Output and place Break Vector 2D.
            H. Connect X to Screen X and Y to Screen Y.
        d. Set Font to your imported font.
        e. Set Scale to 1.5.
    iv. Drag the Output of Draw Text and place Add Hit Box.
        a. Drag the Output of the Vector 2D-Vector 2D of ButtonScreenLocation and Make Vector 2D into Position.
        b. Place a Make Vector 2D with an X of 256 and Y of 128.
            A. Connect the Return Value to Size.
        c. Drag Hitbox Name of Draw Button into Name.

v. Compile and Save.

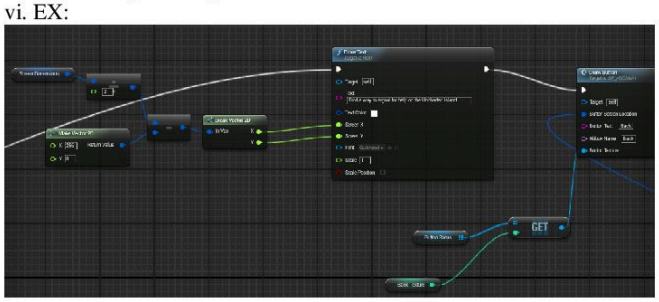
3. Go back to the Event Graph.
   i. Create the following variables.
      a. bool called PlayPressed.
      b. bool called InstructionPressed.
      c. bool called QuitPressed.
      d. bool called BackPressed.
      e. int called PlayTexture.
      f. int called InstructionTexture.
      g. int called QuitTexture.
      h. int called BackTexture.
      i. Vector2D called ScreenDimensions.
      j. Texture2D array called ButtonStatus.
         A. Press the box next to Variable Type to create an array.
         B. Compile and Save.
         C. In Default Value, create 3 elements.
         D. Place Default in 0, Hover in 1, and Active in 2.
   ii. Right Click and place Event Receive Draw HUD.
      a. Drag the Output and place a Set of ScreenDimensions.
         A. Right Click and place a Make Vector 2D. Drag the Return Value into the Screen Dimensions of Set.
         B. Drag Size X into X and Size Y into Y from the Event Receive Draw HUD.
      b. Drag the Output of Set and place Draw Texture.
         A. Set Texture to the title or logo of the game.
         B. Create a Break Vector 2D and connect X and Y to Screen X and Y.
         C. Use the Get of ScreenDimensions and some type of offset as the In Vec for Break Vector 2D.
            1. I Divided Screen Dimensions by a float of 20.
         D. Set Screen W and H to 256.
         E. Set Texture UWidth and VHeight to 1.
         F. Make sure Scale is 1.
      c. Drag the Output of Draw Texture and place a Branch.
         A. Place a Get of InstructionPressed and connect it to Condition.
   iii. EX:



4. Drag the False of Branch and place DrawButton.
    i. Place a Get of ScreenDimensions.
        a. Drag and place Vector2D/Float. Set the Float to 2.
        b. Drag the Output and place Vector2D – Vector2D.
            A. Right Click and place a Make Vector 2D.
            B. Set X to 0 and Y to 64.
            C. Connect the Return Value to the open slot of Vector2D – Vector2D.
        c. Drag the Output into Button Screen Location.
        d. Set Button Text and Hitbox Name to Play.
        e. Place a Get of Button Status.
        f. Drag the Output and place a GET.
            A. Set the int value to the Get PlayTexture.
        g. Connect the Output of GET into Button Texture.
    ii. Drag the Output of Draw Button and place DrawButton.
        a. Drag the Output of the previous Vector2D/Float and place Vector2D + Vector2D.
            A. Right Click and place Make Vector 2D.
            B. Set Y to 64.
            C. Connect the Return Value to Vector2D + Vector2D.
        b. Connect the Output to Button Screen Location.
        c. Set Button Text and Hitbox Name to Instructions.
        d. Place Get of Button Status.
        e. Drag the Output and place a GET.
            A. Set the int value to the Get of InstructionTexture.
        f. Drag the Output into Button Texture.
    iii. Drag the Output of Draw Button and place DrawButton.
        a. Get the Output of the previous Vector2D + Vector2D and place another Vector2D + Vector2D.
            A. Right Click and place a Make Vector 2D.
            B. Set Y to 128.
            C. Connect the Return Value to the open slot of Vector2D + Vector2D.
        b. Drag the Output into Button Screen Location.
        c. Set Button Text and Hitbox Name to Quit Game.
        d. Place Get of Button Status.
        e. Drag the Output and place a GET.
            A. Set the int value to the Get of QuitTexture.
        f. Drag the Output into Button Texture.
    iv. EX:

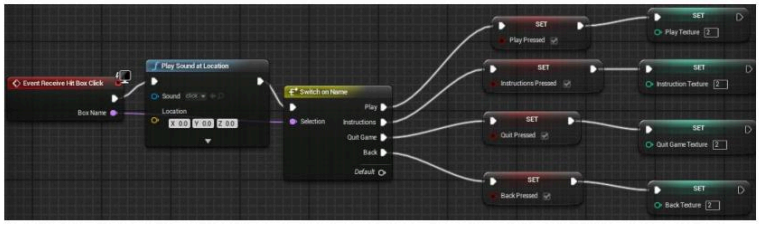


5. Drag the True of Branch and place Draw Text.
    i. Place a Get of ScreenDimensions.
        a. Drag and place Vector2D/Float. Set the Float to 2.
        b. Drag the Output and place Vector2D – Vector2D.
            A. Right Click and place a Make Vector 2D.
            B. Set X to 256 and Y to 0.
            C. Connect the Return Value to the open slot of Vector2D – Vector2D.
        c. Drag the Output and place a Break Vector 2D.
        d. Connect X to Screen X and Y to Screen Y.
    ii. Set Text to: Find a way to signal for help on the Uncharted Island.
    iii. Set Texture Color to White.
    iv. Set your Font.
    v. Drag the Output of Draw Text and place Draw Button.
        a. Find the Vector2D + Vector2D Output used for the Quit Game Button. Connect it to the current Button Screen Location.
        b. Set Button Text and Hitbox Name to Back.
        c. Place Get of Button Status.
        d. Drag the Output and place a GET.
            A. Set the int value to the Get of BackTexture.
        e. Drag the Output into Button Texture.
    vi. EX:

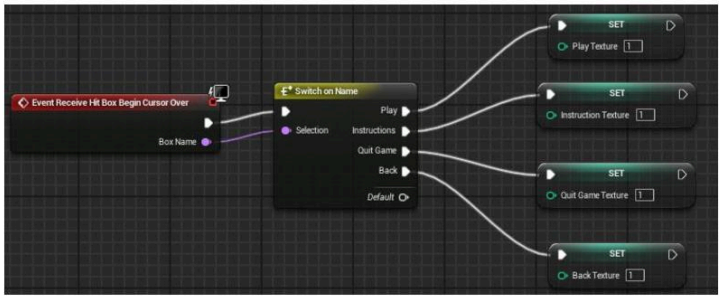


6. Right Click and place Event Receive Hit Box Click.
    i. Drag the Output arrow and place Play Sound at Location.
        a. Set Sound to click.
    i. Search and place Switch on Name in the Find a Node section.
        a. Connect the Output arrow of Click to the Input of Play Sound at Location.
        b. Connect Box Name of Hit Box Click to Selection.
        c. Select Switch and create 4 elements for Pin Names.
            A. 0 is Play, 1 is Instructions, 2 is Quit Game, and 3 is Back.
            B. Make sure the above values are the same as your Hitbox Names from DrawButton.
    ii. Drag Play and place a Set of PlayPressed.
        a. Make sure Play Pressed is checked.
        b. Drag the Output and place a Set of PlayTexture with a value of 2.

iii. Drag Instructions and place a Set of InstructionPressed.
a. Make sure Instruction Pressed is checked.
b. Drag the Output and place a Set of InstructionTexture with a value of 2.
iv. Drag Quit Game and place a Set of QuitPressed.
a. Make sure Quit Pressed is checked.
b. Drag the Output and place a Set of QuitTexture with a value of 2.
v. Drag Back and place a Set of BackPressed.
a. Make sure Back Pressed is checked.
b. Drag the Output and place a Set of BackTexture with a value of 2.
vi. EX:

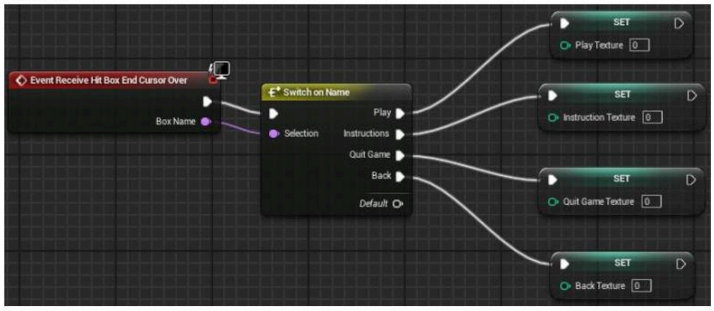


7. Right Click and place Event Receive Hit Box Begin Cursor Over.
i. Copy and paste the Switch on Name with the proper Outputs.
a. Connect the Output arrow of Click to the Input of Switch.
b. Connect Box Name to Selection.
ii. Drag Play and place a Set of PlayTexture with a value of 1.
iii. Drag Instructions and place a Set of InstructionTexture with a value of 1.
iv. Drag Quit Game and place a Set of QuitTexture with a value of 1.
v. Drag Back and place a Set of BackTexture with a value of 1.
vi. EX:

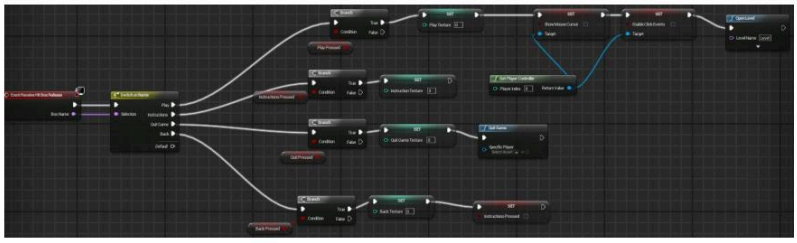


8. Right Click and place Event Receive Hit Box End Cursor Over.
i. Copy and paste the Switch on Name with the proper Outputs.
a. Connect the Output arrow of Click to the Input of Switch.
b. Connect Box Name to Selection.
ii. Drag Play and place a Set of PlayTexture with a value of 0.
iii. Drag Instructions and place a Set of InstructionTexture with a value of 0.

iv. Drag Quit Game and place a Set of QuitTexture with a value of 0.
v. Drag Back and place a Set of BackTexture with a value of 0.
vi. EX:



8. Right Click and place Event Receive Hit Box Release.
    i. Copy and paste the Switch on Name with the proper Outputs.
        a. Connect the Output arrow of Click to the Input of Switch.
        b. Connect Box Name to Selection.
    ii. Drag Play and place a Branch.
        a. Place a Get of PlayPressed and connect it to the Condition.
        b. Drag True and place a Set of PlayTexture with a value of 0.
        c. Right Click and place Get Player Controller.
            A. Drag the Return Value and place a Set of Show Mouse Cursor.
            B. Make sure it is unchecked.
            C. Drag the Return Value and place a Set of Enable Click Events.
            D. Make sure it is unchecked.
        d. Connect the Output of Set PlayTexture to the Input of Set Show Mouse Cursor.
        e. Connect the Output of Set Show Mouse Cursor to the Input of Set Enable Click Events.
        f. Drag the Output and place Open Level.
            A. Set Level Name to the name of the previous levels umap file.
    iii. Drag Instructions and place a Branch.
        a. Place a Get of InstructionPressed and connect it to the Condition.
        b. Drag True and place a Set of InstructionTexture with a value of 0.
    iv. Drag Quit Game and place a Branch.
        a. Place a Get of QuitPressed and connect it to the Condition.
        b. Drag True and place a Set of QuitTexture with a value of 0.
        f. Drag the Output and place Quit Game.
    v. Drag Back and place a Branch.
        a. Place a Get of BackPressed and connect it to the Condition.
        b. Drag True and place a Set of BackTexture with a value of 0.
        c. Drag the Output and place a Set of InstructionPressed.
            A. Make sure it is unchecked.
    vi. EX:



9. Compile and Save. On play, you should now see the menu along with the texture changes. Play should take you to the next level, Instructions should show the explanation and Back, Back should take the player back to the previous screen, and Quit Game should exit from the game.

Result

The player should see a working menu that hints to gameplay and gives them the option to play or leave.

Submission

Packaged projects will be checked in class on Monday (12/1/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. A HUD tutorial (Parts were used for the exercise):
Part 1: https://www.youtube.com/watch?v=7gwgU0UPENA
Part 2: https://www.youtube.com/watch?v=s423ydn3t_E

2. A Menu tutorial using meshes: https://www.youtube.com/watch?v=vBADl-gbITQ

3. You can also install this plug in for UI:
Download: http://coherent-labs.com/download-unreal-engine-4/
First in tutorial series: https://www.youtube.com/watch?v=M-WhfJWEgJo

CGT 345

Due Monday (12/8/14)

Goal

Create a loading and win game screen.

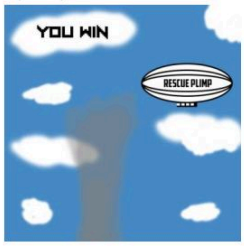
Setup

1. Create a copy of you previous project.

Screen Assets

1. Create a loading screen image.
    i. Should be at least 512x512.
    ii. EX:

2. Create a win game screen image.
    i. Should at least be 512x512.
    ii. EX:



3. Open the Editor and import the Load and Win screen images in the Textures folder.

4. Open the Materials folder.
    i. Create a M_Load material.
        a. Open and change the Blend Mode to Translucent and the Lighting Model to Unlit.
        b. Place the Load Screen texture and connect the top Output to Emissive Color.
        c. Right Click and place a ScalarParameter called OpacityValue. Give it a Default Value of 1.0.
        d. Connect the Output to Opacity.
        e. Save.
    ii. Create a M_Win material.
        a. Open and change the Blend Mode to Translucent and the Lighting Model to Unlit.
        b. Place the Win Screen texture and connect the top Output to Emissive Color.
        c. Save.

5. Find a main menu song, an in game song, and a win song to be imported into the Sounds folder. Double click to edit the settings.
    i. Make sure Looping is checked for the in game and main menu song.
    ii. Make sure Looping is unchecked for win song.

Loading

1. Go open your MainMenu level.

2. Open BP_Check in the Blueprints folder.
    i. Create a new Function called PlayTheme.
        a. Create a new Input called bIsOn.
    ii. Compile and Save.

3. Go to the Blueprints folder and open BP_HUDMain.
    i. Go to Components.
        a. Go to Add Component and create an Audio Component called MenuTrack.
            A. Set Sound to your Main Menu Song.
            B. Set Volume Multiplier to .3.
            C. Uncheck Allow Spacialization and Auto Activate
    ii. Go to Graph.
        a. Go to the Construction Script tab.
        b. Drag the Construction Script Output and place Create Dynamic Material Instance.
            A. Set the Parent to M_Load.
        c. Right Click the Return Value and Promote to Variable. Name it LoadingScreen.

4. Go back to the Event Graph.
    i. Create a bool called bIsLoading.

ii. Drag the Output of the Draw Button for Quit Game and place a Branch.
    a. Place a Get of bIsLoading and connect it to Condition.
iii. Drag True and place Draw Material Simple.
    a. Place a Get of LoadingScreen and connect it to Material.
    b. Place a Get of ScreenDimensions.
    c. Drag the Return Value and place Break Vector 2D.
    d. Connect X to Screen W and Y to Screen H.

5. Select Blueprint Props and BP_Check_C to Interfaces.
    i. Right Click and place Event Play Theme.
    ii. Place a Get of MenuTrack.
        a. Drag the Output and place Play.
    iii. Connect the Output of Play Theme to the Input of Play.

6. Go back down to Event Receive Hit Box Release.
    i. In the Open path, disconnect Open Level.
    ii. Drag the Output of Set for Enable Click Events and place a Set of bIsLoading.
        a. Make sure IsLoading is checked.
    iii. Drag the Output of bIsLoading and place Fade Out.
        a. Set Fade Duration to 1.
    iv. Type timeline in Find a Node and place add Timeline. Call it FadeMenu.
        a. Connect the Output of Fade Out into Play from Start of FadeMenu.
        b. Double click FadeMenu to open.
        c. Add a Float Track called FadeLoad.
        d. Set one key with a value of 0,0 and another key with a value of 2,1.
        e. Set the Length to 2 and have Use Last Keyframe? checked.
    v. Drag the Output and place Set Scalar Parameter Value.
        a. Connect Update of FadeMenu to the Input.
        b. Set the Parameter Name to OpacityValue.
        c. Connect the FadeLoad Output to Value.
    vi. Drag the Finished and place Delay. Give it a Duration of 3.
    vii. Drag Completed and connect it to the disconnected Open Level.
    viii. Compile and Save.

7. Open FPSHUD.
    i. Go to the Construction Script tab.
    ii. Drag the Construction Script Output and place Create Dynamic Material Instance.
        a. Set the Parent to M_Load.
    iii. Right Click the Return Value and Promote to Variable. Name it LoadScreen.
    iv. Create a Vector2D called ScreenDimensions.
    v. Disconnect the first node connected to Event Receive Draw HUD.
    vi. Place a Set of ScreenDimensions.
        a. Right Click and place a Make Vector 2D.
        b. Connect Size X and Size Y of Event into X and Y of Make Vector 2D.
        c. Connect the Return Value to Screen Dimensions of Set.
    vii. Drag the Output of Set into the Input of the disconnected node.

viii. Create a float called FadeValue.

8. Create a Custom Event called LoadEvent.
    i. Drag and place Draw Material Texture.
        a. Place a Get of LoadScreen and connect it to Material.
        b. Place a Get of ScreenDimensions.
            A. Drag the Output and place a Break Vector 2D.
            B. Connect X to Screen W and Y to Screen H.
    ii. Right Click and place Get Player Controller.
        a. Drag the Return Value and place Set Ignore Look Input.
            A. Drag New Look Input and place Float > Float.
            B. Place a Get of FadeValue and connect it to the top slot.
            C. Give the second slot a value of .85.
        b. Drag the Return Value and place Set Ignore Move Input.
            A. Connect the Output of Float > Float to New Move Input.
    iii. Drag the Output of Draw Material Simple and connect it to Input of Set Ignore Look Input.
    iv. Drag the Output of Set Ignore Look Input to Set Ignore Move Input.

9. Place Event Begin Play.
    i. Place add Timeline in Find a Node and name it FadeOut.
        a. Connect Event to Play from Start.
        b. Open FadeOut and Add Float track called FadeStart.
        c. Set the first key to 0,1 and the second key to 5,0.
    ii. Drag Update and place Set of FadeValue.
        a. Set FadeStart to FadeValue.
    iii. Place a Get of LoadScreen.
        a. Drag and place Set Scalar Parameter Value.
            A. Set Parameter Name to OpacityValue.
            B. Place a Get of FadeValue into Value.
    iv. Connect Set FadeValue into the Input of Set Scalar Parameter Value.

10. Place a Custom Event called WinScreen
    i. Create a bool called bHasWon.
    ii. Drag the Output and place a Branch.
        a. Place a Get of bHasWon and connect it to Condition.
    iii. Drag True and place Draw Material Simple.
        a. Set Material to M_Win.
        b. Place a Get of ScreenDimensions.
            A. Drag and place a Break Vector 2D.
            B. Connect X to Screen W and Y to Screen H.
    iv. Drag the Output and place Delay. Give Duration a value of 9.
    v. Drag Completed and place an Open Level with a LevelName of MainMenu.
        a. Set the value of LevelName to your main menu level.

11. Go to the end of the Event Receive Draw HUD chain.

i. Drag the end Output and place a Call Function of WinScreen.
ii. Drag the Output and place a Call Function of LoadEvent.
ii. Make sure WinScreen is drawn before LoadEvent.

12. Compile and Save. Open the Level Blueprint.
    i. Drag the Output of Set View Target Blend and place the Interface Message of Play Theme.
        a. Right Click and place Get Player Controller.
        b. Drag Return and place Get HUD.
        c. Connect the Return Value to Target.

13. Compile and Save. On play, you will see the Load Screen when entering the main level.

Win Screen

1. Open BP_Check.
    i. Create a new Function called Survived.
    ii. Compile and Save.

2. Open BP_Campfire.
    i. Go to the end of the OnComponentBeginOverlap True chain, the one with Spawn Emitter, and drag the last Output to place the Interface Message of Survived.
        a. Right Click and place Get Player Controller.
        b. Drag the Return Value and place Get HUD.
        c. Drag the Return Value into Target.
    ii. Drag the Output of Survived and place a Delay with a Duration of 4.
    iii. Place a Get of Play Fire.
        a. Drag the Output and place Stop.
    iv. Connect Completed to the Input of Stop.
    ii. Compile and Save.

3. Open FPSHUD.
    i. Go to Components and add two Audio Components.
        a. Name the first LevelSound.
            A. Set Sound to In game Song.
            B. Set Volume Multiplier to .3.
            C. Uncheck Allow Spatialization and Auto Activate.
        b. Name the second Winner.
            A. Set Sound to Win Song.
            B. Set Volume Multiplier to .3.
            C. Uncheck Allow Spatialization and Auto Activate.
    ii. Go to Graph.
    iii. Place Event Survived.
    iv. Drag and place a Delay.
        a. Set Duration to 2.
    iii. Search add Timeline in Find a Node and name it LoadWinner.

a. Connect Completed of Delay into Play from Start.
b. Open LoadWinner and add Float Track called Loading.
c. Set one key to 0,0 and the second key to 3,1.

iv. Drag Update and place a Set of FadeValue.
a. Connect Loading to Fade Value.

v. Place a Get of LoadScreen.
a. Drag and place a Set Scalar Parameter Value.
b. Set Parameter Name to OpacityValue.
c. Place a Get of FadeValue and connect it to Value.
d. Connect the Output of Set to the Input of Set Scalar Parameter Value.

vi. Right Click and place Get Player Controller.
a. Drag the Return Value and place Set Ignore Look Input.
A. Make sure New Look Input is checked.
b. Drag the Return Value and place Set Ignore Move Input.
A. Make sure New Move Input is checked.

vii. Connect Set Scalar Output to the Input of Set Ignore Look Input.
viii. Connect Set Ignore Look Output to Set Ignore Move Input.

4. Create a Custom Event called Survivor.
i. Right Click and place a Get of Winner.
i. Drag the Output and place Play.
ii. Drag the Output of Play to the Reverse of LoadWinner.
iii. Drag Finished and place a Branch.
a. Place a Get of bHasWon and connect it to Condition.
iv. Drag False and place a Set of bHasWon.
a. Make sure Has Won is checked.
v. Drag the Output and place a Delay with the Duration of 2.
vi. Drag Completed and place the Call Function of Survivor.
vii. Compile and Save.

Level

i. Go File > Open and open the main level.

2. Go to Level Blueprint.
i. Go to the OnActorBeginOverlap chain for ShackStart.
ii. Drag the Output of Play and place the Interface Message of Play Theme.
a. Right Click and place Get Player Controller.
b. Drag the Return Value and place Get HUD.
c. Drag the Return Value into Target.
d. Make sure Is On is unchecked.
iii. Go to the OnActorEndOverlap chain for ShackStart.
a. Drag the Output of Stop at the end and place a Branch.
A. Right Click and place a Integer >= Integer.
B. Place a Get of Down Count for the top slot and 3 for the bottom.
b. Place a False and place the Interface Message of Play Theme.

A. Right Click and place Get Player Controller.
B. Drag the Return Value and place Get HUD.
C. Drag the Return Value into Target.
D. Make sure Is On is checked.

iv. Go to the Chain that Spawns a Battery after DownCount is greater than 3.
a. Drag the last Output and place a Delay.
A. Set the Duration to the length of your win song.
b. Drag Completed and place an Interface Message of Play Theme.
A. Right Click and place Get Player Controller.
B. Drag the Return Value and place Get HUD.
C. Drag the Return Value into Target.
D. Make sure Is On is checked.

v. Compile and Save.

Result

The game will now have a load screen between the levels and a win screen when certain tasks are completed.

Submission

Packaged projects will be checked in class on Monday (12/8/14).

Building a standalone exe

Make sure your first level is set in Edit > Project Settings > Maps and Modes > Game Default Map. This will be the default scene unless you have a script that leads from one scene to another. The project will package at full settings unless the optimal scalability is coded into the project. The scalable settings in Quick Settings > Engine Scalability will only affect the editor and not the package. Your package must run on a Windows 7 PC (any lab computer in Knoy) to count for credit.

Make sure to include all folders or it will not run

Extra Help:

1. Loading Solution: https://forums.unrealengine.com/showthread.php?2786-Loading-Screen&highlight=loading+screen

2. Cutscene: http://www.youtube.com/watch?v=mrCEtE5RnHw